EDITIONS DU P.S.I.

la pratique du ZX SPECTRUM et du Timex 2000

2. PROGRAMMATION EN LANGAGE MACHINE.

marcel henrot

La pratique du ZX SPECTRUM

Tome 2

COLLECTION LANGAGES

Dictionnaire du Basic — David A. Lien, traduit par Yves Leclerc
Langages de programmation — Stéphane Berche et Claude Lhermite
LOGIC-BASIC — Michel Massiou et Patrick Sénicourt
Programmer en APL — Daniel-Jean David
Programmer en Assembleur — Alain Pinaud
Programmer en Basic — Michel Plouin
Le Basic et ses Fichiers — tomes 1 et 2 — Jacques Boisgontier
Programmer en Fortran — Daniel-Jean David
Programmer en LSE — Stéphane Berche et Yves Noyelle
Programmer en Pascal — Daniel-Jean David et Jean-Luc Deschamps
Comment programmer — Jean-Claude Barbance

COLLECTION MATÉRIELS

La découverte de l'Applesoft — tome 1 — Frédéric Levy et Dominique Schraen
La découverte de l'Applesoft — tome 2 — Frédéric Levy
La Pratique de l'Apple II — volumes I et II — Nicole Bréaud Pouliquen
La pratique de l'Apple II — volume III — Nicole Bréaud Pouliquen et Daniel-Jean David
La découverte de l'ATARI — Daniel-Jean David
La découverte du FX-702 P — Jean-Pierre Richard
La découverte du Goupil — Jean-Yves Michel
Programmer HP-41 — Philippe Descamps et Jean-Jacques Dhénin
Un ordinateur personnel signé IBM — Pierre Longie-Laurent
La pratique du MZ-80 A et K — Jean-Pierre Lhoir
La découverte du PB-100 — Pierrick Moigneau
La découverte du PC-1211 — Jean-Pierre Richard
La découverte du PC-1500 — Jean-Pierre Richard
La découverte du CBM — Daniel-Jean David
La pratique du CBM — volumes I et II — Daniel-Jean David
La pratique du TRS-80 — volumes I, II et III — Pierre Giraud et Alain Pinaud
Exercices pour TRS-80 — Frédéric Lévy
Les graphiques sur TRS-80 — Don Inman traduit par Alain Pinaud
La découverte de la TI-57 — Xavier de la Tullaye
La découverte de la TI-99/4A — Frédéric Lévy
Exercices pour TI-99/4A — Frédéric Lévy
La découverte du VIC — Daniel-Jean David
La pratique du VIC — tome 1 — Daniel-Jean David
La pratique du ZX-81 — tome 1 — Xavier Linant de Bellefonds
La pratique du ZX-81 — tome 2 — Marcel Henrot
Le petit livre du ZX-81 — Trévor Poms, traduit par Ghislaine Lapeyre
La pratique du ZX-Spectrum — tome 1 — Xavier Linant de Bellefonds
La pratique du ZX-Spectrum — tome 2 — Marcel Henrot

COLLECTION GUIDES PRATIQUES

L'APL sur TRS-80 — Claude Nowakowski
CP/M pas à pas — Alain Pinaud
LISP sur Apple II — Nicole Bréaud Pouliquen
La réalisation des programmes — Michel Benelfoul
Méthodes de calcul numérique — tome 1 — Claude Nowakowski
Visicalc sur Apple II — Hervé Thiriez
Visicalc sur TRS-80 — Hervé Thiriez
Pascal sur TRS-80 — Claude Nowakowski

COLLECTION PROGRAMMES

Collèges, poquettes et maths — Jacques Deconchat
Études pour ZX-81 — tomes 1 et 2 — Jean-François Séhan
Jeux, trucs et comptes pour PET/CBM — Michel Benelfoul
Jeux, trucs et comptes pour TRS-80 — Michel Benelfoul et Denis Chauvin
Mathématiques et statistiques — Hervé haut
Modèle pratique de décision — tomes 1 et 2 — Jacques Deconchat
Le TI-99/4A à l'affiche — Jean-François Séhan
Variations pour PC-1211 — Jean-François Séhan
Suites pour PC-1500 — Jean-François Séhan

COLLECTION MEMENTOS

Clefs pour l'Apple II — Nicole Bréaud Pouliquen
Clefs pour le CBM — Daniel-Jean David
CPIM mot par mot — Yvon Dargery
Clefs pour le VIC — Daniel-Jean David
Le Basic de A à Z — Jacques Boisgontier

COLLECTION UTILISATIONS DE L'ORDINATEUR

Le Basic et l'école — tomes 1 et 2 — Jacques Gouet
La comptabilité sur Apple II — Gérard et Serge Lillio
Les finances familiales — Jean-Claude Barbance
Outils financiers et comptables pour l'entreprise — Bernard Sulmon

INITIATION

Visa pour l'informatique — Jean-Michel Jégo
Mon ordinateur — Jean-Claude Barbance
L'ordinateur individuel — Yves Leclerc (en coédition avec S.C.E. Canada)

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'article 40).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

© Éditions du P.S.I., Parc Industriel Nord, Bâtiment 9, 77200 Torcy
1983

Imprimé en France

La pratique du ZX SPECTRUM

par

Marcel Henrot

Tome 2
Programmation en langage machine

Editions du P.S.I.
1983

Ce livre a été réalisé avec la collaboration technique de Xavier Linant de Bellefond, auteur de " La Pratique du ZX-Spectrum " tome 1 Basic approfondi, initiation au langage machine.

SOMMAIRE

		Page
CHAPITRE I	**INTRODUCTION ET RAPPELS**	7
	Les systèmes de numération	8
	Les opérations	11
CHAPITRE II	**LES OPERATIONS DE BASE**	23
	Les mnémoniques de chargement LD	25
	Les codes machine réalisant simultanément plusieurs opérations de base	36
CHAPITRE III	**BOUCLES TESTS ET OPERATIONS COMPLEXES**	41
	Les sauts et les boucles	41
	Les codes machine réalisant les opérations logiques	47
	Les tests et comparaisons	51
	Les rotations et glissements	57
	Utilisation de la pile et des échanges	59
CHAPITRE IV	**TECHNIQUES SIMPLES APPLIQUEES A LA COULEUR ET AU SON**	65
	La couleur	65
	Le son	68

CHAPITRE V	**TECHNIQUES AVANCEES APPLIQUANT LE CLAVIER ET L'AFFICHAGE**	79
	Le clavier	
	L'affichage	85
	Un exemple synthétique : l'impression de caractères agrandis	98
CHAPITRE VI	**L'ANIMATION**	109
	Les éléments de l'animation : hasard et déplacement	109
	Construction d'un jeu d'animation	114
	Variation de la vitesse	120
CHAPITRE VII	**LE MONITEUR**	133
	Les points d'entrée	133
	Routines démontées	138
ANNEXE I	Liste alphabétique de mnémonique	150
ANNEXE II	Conversion hexadécimale-décimale et nombres négatifs (déplacement)	164

CHAPITRE I

INTRODUCTION ET RAPPELS

La programmation, en langage machine, est la suite naturelle de la programmation en Basic. La connaissance du Basic approfondi, qui fait l'objet du tome I de cette collection, ainsi que l'étude attentive du manuel Sinclair, constituent un préalable recommandé à l'apprentissage du langage machine.

La méthode, adoptée dans ce livre, consiste à transposer progressivement, mais complètement, le langage Basic en langage machine ; le lecteur, déjà familiarisé avec le premier, accède ainsi avec plus de facilité à la connaissance de l'autre, qu'en partant de bases entièrement différentes. Une fois acquise cette technique, la programmation en Assembleur de presque tous les ordinateur construits autour du microprocesseur Z80 ne fait pas de difficulté : nous renvoyons, pour ce sujet, au livre de M. Alain Pinaud : "Programmer en Assembleur" dans la même collection.

Parmi les avantages de la programmation machine, citons les trois principaux :

- la vitesse d'exécution est de loin supérieure à celle du Basic, ce qui est primordial pour beaucoup de programmes d'animation ou de gestion ;
- la consommation d'emplacements mémoire est moindre qu'en Basic. A performances égales, les programmes nécessitent moins de place en langage machine ;
- la création de nouvelles commandes, orientées vers les besoins spécifiques de chaque utilisateur, est possible.

Quelques rappels sont nécessaires avant de commencer cette étude : ils portent principalement sur les systèmes de numération dans lesquels s'effectuent les opérations en langage-machine ; sur la manière dont ces opérations arithmétiques ou logiques sont menées par l'ordinateur ; enfin, sur l'organisation générale du système ZX-Spectrum et celle du microprocesseur Z80.

LES SYSTEMES DE NUMERATION

L'ordinateur est une machine qui ne comprend, ne stocke et ne réalise des opérations que sur des informations qui n'ont que deux états logiques possibles : l'état Ø ou l'état 1, bas ou haut, faux ou vrai. Il résulte de ceci que le système binaire est le système de numération le plus adéquat pour traiter ces informations. Le système binaire est à la base d'un autre système, le système hexadécimal qui est très utile pour représenter le contenu des octets de mémoire d'un ordinateur.

Le système binaire

Dès à présent, nous emploierons le mot bit pour désigner un chiffre binaire, le mot quartet pour représenter 4 bits et octet pour 8 bits.

Le système binaire, comme le décimal ou l'hexadécimal, est un système *positionnel*. La position d'un chiffre dans un nombre, à gauche ou à droite de la virgule, confère, à ce chiffre, une certaine valeur, un certain poids. Un système de numération positionnel se représente comme suit :

$$\ldots + a_3 B^3 + a_2 B^2 + a_1 B^1 + a_0 B^0 , + a_{-1} B^{-1} + a_{-2} B^{-2} + \ldots$$

où B est la base, 2 dans le binaire, et les coefficients a sont pris parmi les chiffres du système, 0 et 1 dans le binaire.

Par exemple, le nombre 10011001b peut s'écrire :

$$1.2^7 + 0 + 0 + 1.2^4 + 1.2^3 + 0 + 0 + 1.2^0$$
$$= 128 + 16 + 8 + 1 = 153d$$

(nous ferons suivre un nombre binaire par b, un décimal par d et un hexadécimal par h, sauf quand il n'y a aucune confusion possible).

Le nombre ci-avant peut être mis dans un octet qui est la dimension de la cellule mémoire du ZX-Spectrum.

7	6	5	4	3	2	1	0	poids (exposant de 2)
1	0	0	1	1	0	0	1	nombre

La numération hexadécimale

La numération hexadécimale, de base 16, possède 16 chiffres : 0 à 9, A, B, C, D, E et F (les 6 derniers considérés comme ayant les valeurs 10, 11, 12, 13, 14 et 15). Cette numération est plus dense que la binaire, nous l'employons presque constamment.

En examinant le **tableau 1**, vous voyez qu'un quartet nécessite un chiffre hexadécimal. Donc, chaque quartet d'un nombre binaire se représente par un chiffre hexadécimal. Par exemple : 10011001b = 99h = $9.16^1 + 9.16^0$ = 144 + 9 = 153d.

Un nombre de 16 bits, qui sera le nombre le plus long que nous traiterons, se représente par 4 chiffres hexa. Ainsi, 0100000000000000b = 4000h = 16384d.

Binaire B=2	*Décimal B=10*	*Hexa B=16*	*Binaire B=2*	*Décimal B=10*	*Hexa B=16*
0	0	0	1010	10	A
1	1	1	1011	11	B
10	2	2	1100	12	C
11	3	3	1101	13	D
100	4	4	1110	14	E
101	5	5	1111	15	F
110	6	6	10000	16	10
111	7	7	10001	17	11
1000	8	8	10010	18	12
1001	9	9	10011	19	13

Tableau 1 - correspondance entre les numérations

La virgule flottante

Le ZX-Spectrum stocke et manipule les variables numériques en virgule flottante, en employant la formule

$$N = m.2^{e-128}$$

où l'exposant **e** et la mantisse **m** sont à préciser. Ici, il y a une option à prendre et Sinclair a décidé de préciser **e** et **m** en 5 octets. Cette option conduit à une précision de 9 à 10 chiffres significatifs, ce qui est bien suffisant dans la plupart des cas.

Le premier octet donne la valeur de l'exposant **e**.

Les 4 octets suivants représentent la valeur de la mantisse avec les conventions suivantes :

- la mantisse est entièrement à droite de la virgule (le point pour les Anglais), à gauche de celle-ci, il y a 0 ;
- dans le système binaire, n'importe quel nombre significatif commence par 1. Dans le ZX-Spectrum, le premier sera Ø pour un nombre positif, et 1 pour un négatif.

Ceci amène les règles de conversion suivantes :

- premièrement, écrire les 4 derniers octets en binaire ;
- deuxièmement, si le 1er bit est Ø, mettre 1 à la place, si le 1er bit est 1, mettre - devant le nombre ;
- troisièmement, multiplier la mantisse par 2^x, x étant e - 128 et e la valeur du 1er octet.

Prenons un exemple, entrez le Basic suivant :

```
10 LET a=2.5
20 LET b=PEEK 23627 + 256*PEEK 23628
30 FOR n=0 TO 5
40 PRINT PEEK (b+n);"  ";
50 NEXT n
```

Après exécution, noua avons sur l'écran

```
97  130  32  0  0  0
```

Les 5 octets qui nous concernent sont 130 32 0 0 0 ; le 1er octet représente la lettre a.

Procédons à la conversion en 3 étapes :

1- écrivons la mantisse en binaire

```
0,001b
```

2- il s'agit d'un nombre positif, on remplace le ler 0 par 1

0,101b

3- multiplions par $2^{130-128}$ soit 2^2

le nombre sera 0,101 fois 2^2

En binaire, pour multiplier par 2 (la base B), on déplace la virgule d'un rang vers la droite.

le nombre sera 10,1b ou $2^1 + 0, + 2^{-1} = 2,5d$

Remarque - Pour des nombres non fractionnaires, allant de - 65535 à 65535, le XZ-Spectrum les représente en 5 octets, d'une façon particulière qui est expliquée dans le manuel Sinclair. Par exemple, dans le Basic ci-dessus, remplacez :

```
1Ø LET a=23552
```

Après exécution, nous avons

97 0 0 0 92 0

Les 5 octets qui nous concernent sont 0 0 0 92 0.

Le ler octet est toujours 0 ;

Le 2e octet est 0 : nombre positif ;

Les 3e et 4e octets représentent le nombre avec l'octet le moins significatif d'abord, d'où le nombre est :

0101110000000000b = 5C00h = 23552d

LES OPERATIONS

Atithmétiques

Le microprocesseur effectue l'addition et la soustraction. Cette dernière opération se transforme en addition grâce à la représentation des nombres négatifs, appelés nombres avec signe ***(tableau 2)***.

La base étant B, on appelle complément restreint d'un nombre, son complément à B-1 et complément vrai, son complément à B.

Ceci étant défini, pour représenter un nombre avec signe, nous prenons la convention suivante : le bit de poids le plus haut sera un bit de signe "0" pour un nombre positif et "1" pour un négatif. Ensuite, chaque nombre négatif est représenté comme le complément vrai du nombre positif correspondant. Un octet pourra repré-

senter un nombre de -128 à 127 et 2 octets un nombre de -32768 à 32767.

Les registres du microprocesseur contiennent 8 bits, certains peuvent se coupler pour contenir 16 bits. De ce fait, les additions et soustractions se feront modulo 256 dans un cas, et modulo 65536 dans l'autre.

C'est comme si, en décimal, nous ne disposions que de 4 cases, par exemple, les opérations se feraient alors modulo 10000 puisqu'on ne pourrait représenter que des nombres de 0 à 9999. Nous verrons que l'indicateur d'état C signalera si le résultat est modulo.

Binaire B=2	*Décimal B=10*	*Hexa B=16*	*Binaire B=2*	*Décimal B=10*	*Hexa B=16*
10000000	-128	80	00000000	0	0
10000001	-127	81	00000001	1	1
10000010	-126	82	00000010	2	2
10000011	-125	83	"	"	"
"	"	"	"	"	"
"	"	"	01111101	125	7D
11111110	-2	FE	01111110	126	7E
11111111	-1	FF	01111111	127	7F

Tableau 2 - Représentation des nombres avec signe

Logiques

Le microprocesseur réalise différentes opérations logiques. Ces opérations s'effectuent toujours sur 8 bits.

Le complément restreint

C'est le complément à B-1 (B étant la base) effectué bit à bit. Le complément restreint de A s'écrit $\overline{A}$ (lire A barre).

Exemple : si A = 10001000,
alors $\overline{A}$ = 01110111.

Le complément vrai

C'est le complément à B ou, plus simplement, c'est le complément restreint +1. Dans l'exemple précédent, c'est

01111000. Si vous préférez, c'est le résultat de l"opération 0 - A.

L'opération ET

Cette opération s'effectue sur deux opérandes, bit à bit. On écrit A ∧ B (lire A et B).

Exemple :

```
A  10101111
B  00000111
   --------
   00000111
```

Il suffit d'appliquer les règles suivantes pour chaque bit:

```
0 ∧ 0 = 0
0 ∧ 1 = 0
1 ∧ 0 = 0
1 ∧ 1 = 1
```

L'opération OU

Elle s'effectue également bit à bit, sur deux opérandes. On écrit A ∨ B (lire A ou B).

Exemple :

```
A  00110011
B  01000100
   --------
   01110111
```

en appliquant les règles suivantes :

```
0 ∨ 0 = 0
0 ∨ 1 = 1
1 ∨ 0 = 1
1 ∨ 1 = 1
```

L'opération OU exclusif

Elle s'effectue aussi bit à bit, sur deux opérandes. On écrit A ∀ B (lire A ou exclusif B).

Exemple :

```
A  01010101
B  10100101
   --------
   11110000
```

en appliquant les règles :

```
0 ∀ 0 = 0
0 ∀ 1 = 1
1 ∀ 0 = 1
1 ∀ 1 = 0
```

LE SYSTEME ZX-SPECTRUM

La ***figure 1*** montre les principaux composants du ZX-Spectrum :

- ***le microprocesseur Z80A***, qui tourne à 3,5 MHz, le quart de l'oscillateur à quartz X1. Une période T dure

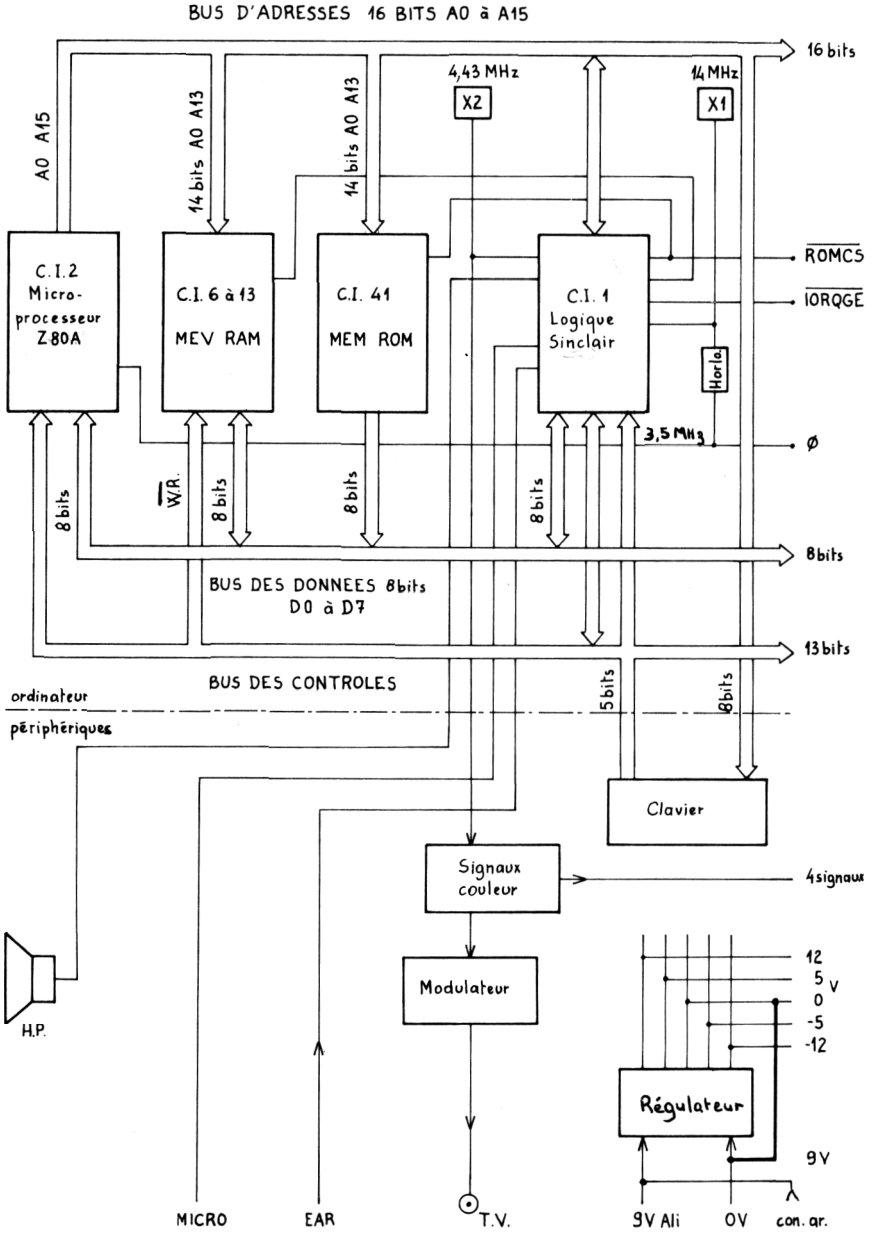


Figure 1 - synoptique du ZX Spectrum

0,2857 microsec. L'instruction la plus courte du Z80A dure 4T, soit 1,143 microsec., et la plus longue 23T, soit 6,571 microsec. ;

- ***la mémoire vive***, qui se compose de 8 C.I. (circuits intégrés) pour former les 16 K d'octets de la configuration de base ;
- ***la mémoire morte***, également de 16 K d'octets, qui contient le système d'exploitation ;
- ***l'ULA*** qui contient toute la logique Sinclair. Ce C.I. sert d'interface aux périphériques ;
- ***les périphériques*** : le *clavier*, le *modulateur TV* et un *petit haut-parleur*. Tous ces composants s'interconnectent par des conducteurs. Certains de ceux-ci s'assemblent pour former des bus. Le bus d'adresses réunit 16 conducteurs A0 à A15. Par simplification, nous employons le mot bit A0 pour le conducteur qui véhicule le bit A0. Le bus des données réunit 8 conducteurs D0 à D7. Le bus des contrôles réunit 13 conducteurs. Chaque contrôle porte un nom différent. Certains contrôles sortent du microprocesseur pour montrer son comportement, d'autres y entrent pour le modifier ou l'interrompre.

La mémoire vive

Celle-ci s'étend sur 16 K d'octets allant de l'adresse 16384 à 32767. Une adresse est un emplacement mémoire portant un numéro. Chaque adresse peut contenir une information de 8 bits. Certaines adresses sont fixes, comme le début du fichier d'affichage ou le début des variables systèmes ; d'autres sont variables, comme le début de la zone programme ou de la zone variables. Pour indiquer le contenu d'une adresse, nous emploierons des parenthèses, ainsi (24000)=80h=128d signifie que le contenu de la cellule mémoire 24000 contient 8 bits qui représentent 128d. De même, des doubles parenthèses signifient : contenu du contenu de. ; ainsi ((VARS))= 61h=97d signifie que le contenu de la variable système VARS donne une adresse x = (VARS) et le contenu de cette adresse donne un octet (x) = 97d.

La mémoire morte

Cette mémoire est implantée aux adresses 0 à 16383. Elle contient le programme moniteur qui permet l'usage de plusieurs périphériques et un interpréteur Basic qui

donne, à l'utilisateur, l'usage d'un langage Basic évolué, contenant de nombreuses commandes et fonctions.

La **MEM** contient beaucoup de routines en codes machine qui sont appelées par l'interpréteur, au fur et à mesure de ses besoins. Nous emploierons certaines de ces routines au cours de l'ouvrage, et nous en donnerons les points d'entrée au moment voulu.

La logique Sinclair ULA

Cette logique, réunie dans un seul C.I., sert d'interface entre l'ordinateur et les périphériques. Certains périphériques sont intégrés dans le boitier : le clavier, le modulateur UHF et un petit haut-parleur (qui ne parle pas si haut que cela). La gestion de micro-disquettes est prévue et les commandes nécessaires à son exploitation figurent parmi les commandes Basic disponibles.

Grâce au connecteur arrière, on peut ajouter tout ce que l'on veut : interface RS 232, manettes de jeu, synthétiseur, convertisseur analogiques-digital, etc...

Enfin, une très importante activité de l'ULA est la gestion de l'écran. L'ULA génère 50 demi-images par seconde et décharge ainsi le microprocesseur de cette tâche qu'il était obligé d'assumer dans le ZX81 et qui nécessitait le mode SLOW.

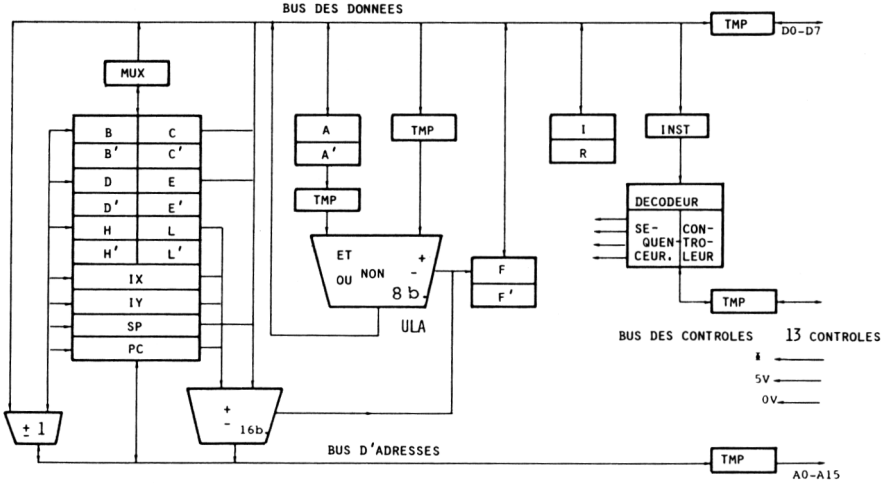


Figure 2 - Organisation interne du microprocesseur Z80A

Le microprocesseur Z80A

La ***figure 2*** donne l'organisation interne de ce microprocesseur.

Disons quelques mots sur ses composants :

- les unités qui effectuent des opérations : un pour incrémenter et décrémenter, un pour les opérations arithmétiques sur 16 bits et un pour les opérations arithmétiques et logiques sur 8 bits ;
- le MUX est un multiplexeur qui sélectionne un des registres situés en dessous de lui ;
- les registres tampons TMP. Prenons, comme exemple, le tampon sur le bus des données. Ce registre de 8 bits a trois états possibles : les 2 états logiques 0 ou 1 et l'état haute impédance. Le microprocesseur le met dans cet état lorsqu'il manipule des données internes. Cet état isole le microprocesseur du monde extérieur ;
- les différents registres : tous ces registres sont disponibles pour l'utilisateur, à l'exception du registre d'instruction INST, des tampons et du multiplexeur. Un registre contient une certaine configuration de bits. Quelle que soit la signification de cet arrangement de bits (instruction, nombre, caractère, attribut couleur, etc...), on peut toujours le considérer comme un nombre binaire pur.

Voyons ces différents registres :

Le registre d'instruction reçoit une instruction, la décode, puis l'exécute en envoyant des signaux aux différents composants du microprocesseur et vers l'extérieur par le bus des contrôles. Le nombre de cycles d'horloge T, requis pour l'exécuter, dépend de l'instruction elle-même.

Le registre I tient l'octet haut d'une adresse pour un système d'interruption vectorisé, relié à de nombreux périphériques (modem, disquette, imprimante, etc...).

Le registre R, incrémenté régulièrement, sert à rafraîchir des zones de mémoires dynamiques.

Les registres B, C, D, E, H et L sont d'usages généraux. Ils servent d'intermédiaires entre la mémoire extérieure et le microprocesseur pour effectuer des opérations, des transferts, etc... Ces registres peuvent se coupler pour former des registres de 16 bits : BC, DE et HL.

Les registres A' à L' doublent les registres A à L. Un seul jeu est employé à la fois. Ils servent de mémoire temporaire.

Les registres d'index IX et IY servent à l'adressage indexé pour pointer sur un emplacement mémoire.

Le registre pointeur de pile SP pointe sur un emplacement mémoire. Dans celle-ci, on réserve une zone qui servira de pile au microprocesseur. Cette pile fonctionne suivant le principe LIFO (Last In First Out) : la dernière information entrée sera la première à sortir. La tête de la pile a une adresse assez haute en mémoire et on empilera aux adresses directement en dessous. Donc, quand on pousse un octet dans la pile, SP sera décrémenté et, quand on le retirera, SP sera incrémenté.
A noter qu'il y a trois zones de piles dans la MEV = une pile pour les GOSUB du Basic, la pile pointée par SP et une pile pour le calculateur. Entre les deux dernières piles, se situe la zone réserve.

Le registre PC pointe sur l'instruction à exécuter. Après exécution, le PC est incrémenté, plusieurs fois si nécessaire, pour pointer sur l'instruction suivante.

Le registre A ou accumulateur se trouve sur une des deux entrées de l'unité opératoire 8 bits. Cette unité effectue une opération sur les deux opérandes présentes à ses deux entrées, ceci amène un emploi fréquent de A.

Le registre F

Le registre F est un registre spécial. Les bits qu'il contient sont des indicateurs d'état. Chaque indicateur est mis à 1 ou à 0, suivant le résultat de l'opération effectuée.

7	6	5	4	3	2	1	0	poids
S	Z	-	H	-	P/V	N	C	bits

Figure 3 - configuration du registre

L'indicateur C est l'indicateur de report (addition) ou de retenue (soustraction). Il est mis à 1 quand le résultat est modulo 256 ou modulo 65536.

L'indicateur P/V a deux fonctions. Une fonction de dépassement V pour les opérations arithmétiques et de parité P pour les opérations logiques. V sera mis à 1 quand il y a dépassement de capacité de calcul sur des nombres avec signe. P sera mis à 1 quand il y a un nombre pair de bits 1 (ou de bits 0).

L'indicateur Z est mis à 1 si le résultat de l'opération est 0.

L'indicateur S indique le signe. S est mis à 1 quand le bit le plus significatif du résultat est 1.

Les indicateurs N et H servent pour des opérations qui portent sur des quartets codés en DCB. Ce mode de codage et les opérations qui en résultent ne sont pas examinés dans ce livre.

Les codes machine

Les codes machine représentent des instructions en langage machine que le microprocesseur comprend. Chaque instruction, reçue par le microprocesseur, lui fait exécuter une ou plusieurs opérations. Chaque instruction sera représentée par un mnémonique. A partir du chapitre 2, chaque mnémonique sera expliqué, en donnant l'instruction qu'il représente, son code machine et son influence sur les indicateurs d'état. Toutes ces explications sont reprises en annexe, par ordre alphabétique des mnémoniques.

Un octet peut avoir 256 configurations de bits différentes. En affectant un code machine à chacune de ces configurations, on disposerait de 256 instructions codées en un seul octet. Cependant, le Z80A permet l'emploi de plus de 600 instructions. A cette fin, on emploie les préfixes CB et ED, suivi d'un autre octet. Ce sont des instructions à deux octets. Certaines instructions demandent des nombres (variables ou constantes) pouvant aller jusqu'à 65535, ce qui nécessite un ou deux octets. De ce fait, une instruction pourra comporter de un à quatre octets. Le registre PC sera mis à jour automatiquement. Il existe deux autres préfixes, DD et FD, qui indiquent les instructions portant sur les registres d'index IX et IY.

Prenons un exemple : on veut charger le nombre 5 dans le registre B. L'instruction B=5 aura, pour mnémonique, LD B,5 et, pour codes mahcine, 06,05. Cette instruction nécessite deux octets. La déclaration Basic 10 LET B=5 nécessite 15 octets en zone mémoire et 6 en zone variables, après exécution.

Pour représenter les codes machine, nous emploierons un tableau divisé en 5 colonnes. Une ou plusieurs colonnes peuvent rester vierges.

Adresse	*Codes machine*	*Etiquette*	*Mnémonique*	*Commentaires*
28000	06,05		LD B,5	

Tableau 3 - Présentation des codes machine

La colonne 1 sera écrite en décimal, la colonne 2 en hexadécimal et la colonne 4 dans l'une des numérations, selon le cas.

Les colonnes étiquette et commentaires seront utilisées, si nécessaire. Seules, les deux premières colonnes sont indispensables. Nous devons loger les codes machine de la colonne 2 aux adresses indiquées par la colonne 1.

Nous utilisons la zone mémoire au dessus de RAMTOP parce que le ZX-Spectrum offre des commandes utiles pour le réaliser et que le sauvetage sur cassette et le chargement n'offrent aucune difficulté.

Programme de chargement de codes machine

```
1000 PRINT " Combien d'octets possède votre","programme?"
1010 INPUT FLASH 1;"Entrez le nombre d'octets",a
1020 LET b=32598-a: CLEAR b
1030 LET a=PEEK 23730+256*PEEK 23731+1: PRINT " Votre programme
     commence a","l'adresse  ";a
1040 PRINT ' " Entrez vos codes machine par","groupe de 2"
1045 INPUT FLASH 1;"Entrez 2 chiffres hexa.",a$
1050 CLS: PRINT INVERSE 1;"Adresse";TAB 10;"HEXA"."DEC.":GO TO 1060
1055 INPUT FLASH 1;"Entrez 2 chiffres hexa,",a$
1060 IF a$="l" THEN GO TO 1200
1070 IF a$="e" THEN GO TO 1250
1080 IF a$="s" THEN STOP
1090 IF LEN a$ <> 2 THEN GO TO 1055
1100 IF CODE a$ >96 THEN LET a$(1)=CHR$ (CODE a$(1)-32)
1110 IF CODE a$(2) > 96 THEN LET a$(2)=CHR$ (CODE a$(2)-32)
1120 IF CODE a$ > 70 OR CODE a$(2) > 70 THEN GO TO 1055
1130 LET b=16*CODE a$ -112*(CODE a$ > 57)+ CODE a$(2)
     -7*(CODE a$(2) > 57)-816
1140 POKE a,b
1150 PRINT a;TAB 10;a$,b;"2esp"
1160 LET a=a+1
1170 GO TO 1055
1200 LET b=PEEK a
1210 LET c=INT (b/16)
1220 LET a$=CHR$ (c+48+7*(c> 9))+CHR$ (b-16*c+48+7*((b-16*c) > 9))
1230 GO TO 1150
1250 LET a=a-1
1260 POKE 23689, PEEK 23689 +1
1270 GO TO 1055
```

Les déclarations ont été chiffrées à partir de 1000 pour que vous puissiez mettre votre programme Basic d'exploitation entre 1 et 1000. Après la mise au point, vous pouvez l'effacer ou le laisser mais, dans ce cas, ajoutez la déclaration 999 STOP pour éviter que l'ordinateur exécute ce programme à la suite du vôtre

Ce programme possède les facilités suivantes : après avoir entré le nombre d'octets que vous voulez charger, l'ordinateur vous demandera d'entrer vos codes machine, vous entrez alors deux chiffres hexadécimaux. Si vous entrez 1, 3 ou plus de caractères, il n'arrive rien. Il en sera de même si vous entrez deux chiffres non hexadécimaux, comme g2 ou 8h.

Pour lister votre programme ou une routine de la MEV, vous entrez l, vous avez alors le contenu de l'adresse.

Si vous vous êtes trompé en entrant des codes machine, effacez en entrant e, vous reviendrez à l'octet pénultien.

Enfin, pour arrêter le programme, entrez s.

CHAPITRE II

LES OPÉRATIONS DE BASE

Au titre des opérations de base figure, naturellement, celle qui donne accès à tout programme en langage machine, en effectuant la sortie du programme Basic. C'est l'objet de la fonction USR, à la mise en oeuvre de laquelle sera consacré un premier développement, bien qu'il ne s'agisse pas, à proprement parler, d'une opération de langage-machine.

L'opération symétrique de retour au Basic, à partir d'un programme en langage-machine, est en revanche commandée, elle, par une instruction de ce langage : il en sera parlé immédiatement après.

Les autres opérations de base consistent essentiellement en opérations de chargement et de rangement. Ce sont des transferts entre MEV et registres du microprocesseur. Nous examinons aussi les opérations simples que peuvent subir les registres : l'incrémentation (addition de 1), la décrémentation (soustraction de 1), l'addition et la soustraction de constantes ou du contenu d'un autre registre.

Nous terminons le chapitre en décrivant des instructions puissantes qui effectuent plusieurs opérations ci-dessus, à la fois.

LA FONCTION "USR". LE RETOUR AU BASIC RET.

La fonction **USR** est celle qui donne accès aux programmes en C.M. élaborés par l'utilisateur. Chaque octet de C.M. est introduit dans une cellule mémoire d'une manière séquentielle. Cette cellule mémoire possède une adresse comprise entre 0 et 65535. Donc USR sera suivi par un nombre entier positif plus petit que 65536, et, comme USR est une fonction, elle sera précédée par une commande basic. Par conséquent, tout programme écrit en C.M. et introduit en mémoire devra posséder au moins une ligne basic pour s'exécuter.
Par exemple :

```
PRINT USR 28000
```

exécutera tous les C.M. qui sont à 28000 et seq.

Pour que l'ordinateur, après notre programme, ne continue pas à exécuter des C.M. indésirables, nous devons mettre à la fin de notre programme un C.M. pour le retour au basic. Ce C.M. est C9h-201d-, son mnémonique est RET et il exécute l'instruction qui consiste à retourner à la ligne basic qui contient USR.
Faisons un programme d'un seul C.M.

```
28000 C9      RET
```

en faisant la commande POKE 28000,201

Pourquoi prenons-nous 28000 ? Parçe qu'il n'y a rien dans la mémoire à cet endroit, tout au moins quand on vient de mettre le ZX-Spectrum sous tension.
Faites maintenant la commande

```
PRINT USR 28000
```

et 28000 s'affiche.

Anticipons légèrement sur le paragraphe suivant, où nous expliquerons les mnémoniques LD, et entrons le programme :

```
28000  06,00     LD B,0
28002  0E,0A     LD C,10d
28004  C9        RET
```

en faisant les commandes :

```
POKE  28000,6
POKE  28001,0
POKE  28002,14
POKE  28003,10
POKE  28004,201
```

Maintenant, faites PRINT USR 28000
et 10 s'affiche

Ceci nous fait comprendre le fonctionnement de USR qui se décompose ainsi :
- l'ordinateur place l'argument de USR dans son regis-

tre double BC.
- l'ordinateur exécute la routine
- quand RET survient, il retourne le contenu de son registre double BC

Dans le premier cas, USR a mis 28000 dans BC et RET a retourné 10 le contenu de BC.

Ceci est très important, car le retour peut être exploité dans un programme Basic.

Au lieu de PRINT USR 28000, faites LET a=USR 28000, puis PRINT a: 10 s'affichera. On peut employer presque toutes les commandes Basic, comme pour toute autre fonction, telle que PRINT AT USR ..., RUN USR ..., GO TO USR ..., IF USR ..., PAUSE USR ..., etc...

Si vous n'avez pas besoin du retour pour votre Basic, faîtes RANDOMIZE..., c'est plus économique

LES MNEMONIQUES DE CHARGEMENT LD

Le mnémonique **LD**, qui signifie charger, n'influence pas les indicateurs d'état, à l'exception de **LD A,I** et **LD A,R** qui agissent sur Z et S.

CHARGEMENT DE REGISTRE

Chargement d'un registre par un autre registre

78	LD A,B	LET A=B

qui signifie charger B dans A. Les deux arguments sont séparés par une virgule : c'est toujours ce qui est à gauche de la virgule qui change de valeur.

Comme il y a 7 registres, il existe 49 C.M. pour ces opérations.

Pour les registres doubles, il n'y en a que 3, les voici :

F9	LD SP,HL	LET SP=HL
DD,F9	LD SP,IX	LET SP=IX
FD,F9	LD SP,IY	LET SP=IY

Un mnémonique, tel que LD BC,HL, n'est pas valable. Pour le réaliser, il faut faire :

```
44          LD B,H
4D          LD C,L
```

Chargement d'une constante, de 0 à 255, dans un registre

3E,nn	LD A,n	LET A=n

Les 7 registres simples peuvent être chargés par une constante. Cette constante n est représentée par nn dans les C.M. pour rappeler qu'il faut deux chiffres h. (même pour charger 0, il faut entrer 00).

Chargement d'une constante, de 0 à 65535, dans un registre

01,mm,nn	LD BC,nm	LET BC=nm

Les 3 registres couplés et les 3 registres doubles peuvent être chargés. Dans le Basic, si on considère nm comme deux octets, la correspondance sera :

LET BC=256*n+m
ou bien LET B=n et LET C=m

Attention : vous voyez, dans les C.M., que les deux chiffres h mm représente l'octet bas et nn l'octet haut. Il faut toujours entrer l'octet bas, puis l'octet haut. Par exemple :

01,20,00	LD BC,0020h	LET BC=32
21,00,58	LD HL,5800h	LET HL=22528

car, 58h=88d et comme octet haut 88*256=22528 (adresse du début du fichier d'attributs).

CHARGEMENT PAR UN CONTENU MEMOIRE

Chargement d'un registre par le contenu d'une adresse

0A	LD A,(BC)	LET A=PEEK BC
1A	LD A,(DE)	LET A=PEEK DE
3A,mm,nn	LD A,(nm)	LET A=PEEK nm

BC, DE et nm représentent des adresses en mémoire, et c'est l'octet contenu dans ces cellules adressées qui est chargé dans A. Ces trois instructions ne peuvent se faire qu'avec A. Par contre, les instructions

7E	LD A,(HL)	LET A=PEEK HL
DD,7E,dd	LD A,(IX+d)	LET A=PEEK (IX+d)
FD,7E,dd	LD A,(IY+d)	LET A=PEEK (IY+d)

peuvent se faire avec n'importe quel autre registre simple.

Dans LET A=PEEK (IX+d), les parenthèses sont Basic. Elles sont nécessaires pour le Basic du ZX-Spectrum. Elles n'ont rien à voir avec les parenthèses du mnémonique qui signifient "contenu de".

Dans les registres d'index IX et IY, on a la faculté d'ajouter un déplacement d. Ce déplacement peut être positif ou négatif, car d est considéré, par l'ordinateur, comme un nombre avec signe (voir chapitre 1). Le déplacement est positif quand d a une valeur comprise entre 0 et 7Fh - 127d - et négatif pour d compris entre 80h - 128d - et FFh -255 d -. Dans ce dernier cas, la valeur réelle va de - 128 à - 1.

```
Si IY = 23610          ERR-NR
(IY+17) = (23627)      (VARS)
(IY+252) = (23606)     (CHARS)
```

En C.M., n'employez pas IY, ou bien, employez-le uniquement avec la valeur ci-dessus. En effet, au cours de l'initialisation, quand le Spectrum est mis sous tension, l'ordinateur affecte la valeur 23610 à IY. Ceci permet à l'ordinateur, avec différentes valeurs de d, de faire pointer IY sur n'importe quelle variable système de 23552 à 23733.

Il n'y a que A que l'on puisse charger avec le contenu d'une adresse spécifiée par une constante. LD B,(nm) n'est pas valable, il faut faire :

```
     21,mm,nn              LD HL,nm
     46                    LD B,(HL)
ou bien :
     3A,mm,nn              LD A,(nm)
     47                    LD B,A
```

Vous prendrez l'une ou l'autre manière, selon que A ou HL est disponible à ce moment-là.

Chargement d'un registre double par le contenu de deux adresses successives

```
     ED,4B,mm,nn           LD BC,(nm)
qui signifie
     LET BC =PEEK nm+256*PEEK (nm+1)
ou   LET C=PEEK nm  et  LET B=PEEK (nm+1)
```

```
ED,5B,mm,nn          LD DE,(nm)
ED,6B,mm,nn          LD HL,(nm)
ED,7B,mm,nn          LD SP,(nm)
   2A,mm,nn          LD HL,(nm)
DD,2A,mm,nn          LD IX,(nm)
FD,2A,mm,nn          LD IY,(nm)
```

Il y a deux C.M. possibles pour charger HL. Nous prendrons celui de trois octets (il y a quelques redondances de ce genre parmi le jeu d'instructions du Z80A).

Attention : quand on charge un registre double, on effectue deux opérations :

```
RRbas = (nm)
RRhaut = (nm+1)
```

on charge deux octets, ce que montre bien la correspondance Basic.

Exemple 1 : l'expression Basic

```
PRINT PEEK 23684+256*PEEK 23685
```

donne l'adresse de la position d'écriture PRINT. On emploie la variable système DF-CC ; en C.M., il vient :

```
32550 ED,4B,84,5C     LD BC,(23684)
32554 C9              RET
```

PRINT USR 32550 vous donnera l'adresse du premier octet de la position PRINT.

Pour cet exemple et les suivants, employez le programme de chargement du chapitre 1 et, à la demande "Entrez le nombre d'octets", répondez 49.

Exemple 2 : entrez en Basic

```
PRINT PEEK (PEEK 23684+256*PEEK 23685)
```

pour avoir le contenu du 1er octet de la position PRINT.

```
32550 2A,84,5C        LD HL,(23684)
32553 06,00           LD B,0
32555 4E              LD C,(HL)
32556 C9              RET
```

et PRINT USR 32550 vous donne le résultat.

Dans ces deux exemples, il s'agit de l'adresse du 1er octet (et de son contenu) de la position PRINT. Nous reviendrons plus loin sur le fichier d'affichage du ZX-Spectrum. Ces exemples illustrent les C.M. vus jusqu'ici.

EXERCICE 1 : la variable système S-POSN contient les coordonnées de la position PRINT = n° de colonne en 23688 et n° de ligne en 23689. Ecrivez un programme en C.M. pour connaître le numéro de colonne et le numéro de ligne.

CHARGEMENT DANS LA MEMOIRE

Chargement d'un octet mémoire (contenu d'une adresse) par un registre

02	LD (BC),A	POKE BC,A
12	LD (DE),A	POKE DE,A
32,mm,nn	LD (nm),A	POKE nm,A

Ici aussi, BC, DE et nm représentent des adresses mémoires, mais c'est l'octet de ces cellules qui est chargé par A. Ces trois instructions ne peuvent se faire qu'avec A. Par contre, les instructions

77	LD (HL),A	POKE HL,A
DD,77,dd	LD (IX+d),A	POKE IX+d,A
FD,77,dd	LD (IY+d),A	POKE IY+d,A

peuvent se faire avec n'importe quel registre simple.

Il n'y a que A qui puisse charger dans une adresse spécifiée par une constante. Pour réaliser LD (nm),B, on fait

78	LD A,B
32,mm,nn	LD (nm),A

Chargement d'une constante dans un octet mémoire

36,nn	LD (HL),n	POKE HL,n
DD,36,dd,nn	LD (IX+d),n	POKE IX+d,n
FD,36,dd,nn	LD (IY+d),n	POKE IY+d,n

Dans les C.M. des registres d'index, d vient toujours en 3e position. Si on veut LD (IX),n, on fait d=0.

DD,36,00,nn	LD (IX+0),n

Tous les C.M., concernant les registres d'index, sont identiques, à l'exception du 1er octet : DD pour le registre IX et FD pour IY.

Chargement de deux octets mémoire successifs par un registre double

```
ED,43,mm,nn     LD (nm),BC          qui signifie
          POKE nm,C  et  POKE nm+1,B  ou
         { POKE nm,BC-INT(BC/256)*256
         { POKE nm+1,INT(BC/256)
ED,53,mm,nn     LD (nm),DE
ED,63,mm,nn     LD (nm),HL
22,mm,nn        LD (nm),HL
ED,73,mm,nn     LD (nm),SP
DD,22,mm,nn     LD (nm),IX
FD,22,mm,nn     LD (nm),IY
```

Ici, il y a deux possibilités pour HL.

Attention : ces C.M. réalisent deux instructions :

```
(nm)   = RRbas
(nm+1) = RRhaut
```

comme le montre le Basic correspondant.

Exemple 3 : la commande RAND se réalise comme suit, en Basic :

```
RANDOMIZE 768
```

et en C.M. :

```
32550 21,00,03  LD HL,300h      :768d
32553 22,76,5C  LD (23670),HL   :(SEED)
32556 C9        RET
```

LES CODES MACHINE REALISANT LES OPERATIONS ARITHMETIQUES

LES ADDITIONS

```
C6,nn           ADD A,n         LET A=A+n
80              ADD A,B         LET A=A+B
```

On peut ajouter n'importe quel registre simple à A.

```
86              ADD A,(HL)      LET A=A+PEEK HL
DD,86,dd        ADD A,(IX+d)    LET A=A+PEEK (IX+d)
FD,86,dd        ADD A,(IY+d)    LET A=A+PEEK (IY+d)
```

On ne peut additionner un octet (constante, registre ou mémoire) qu'à A.

Toutes ces instructions influencent les indicateurs S, Z, P/V et C.

C'est le résultat de l'addition qui influence ces indicateurs, ainsi :

```
C6,00          ADD A,0        LET A=A
```

laisse A inchangé, mais C et P/V = 0, et S et Z ne changent pas. C'est la seule instruction, vue jusqu'ici, qui fait $C_i = 0$

Remarque : nous mettons C_i pour désigner l'indicateur C, pour le différencier du registre C.

```
     ADD A,B  veut dire  LET A=A+B
mais aussi   IF (A+B) > 255 THEN LET Ci=1
mais aussi   IF (A+B)=256  THEN LET Z=1
mais aussi   IF (A+B) > 127 THEN LET S=1
mais aussi   IF (A+B) > 127 AND (A+B) < 256 THEN LET P/V=0
```

tout cela parce que les opérations se font modulo 256.

Ceci montre également que la traduction fidèle au Basic sera toujours circonstanciée. Un C.M. peut faire plusieurs commandes Basic, et une commande Basic peut nécessiter plusieurs C.M.

Exemple 4 : addition modulo 256

```
32550 3E,00     LD A,x
32552 C6,00     ADD,A,y        :fait x=y
32554 4F        LD C,A
32555 06,00     LD B,0
32557 C9        RET
```

Faîtes le programme Basic :

```
 10 INPUT "Entrez x",x: POKE 32551,x
 20 INPUT "Entrez y",y: POKE 32553,y
 30 PRINT x;"+";y;"="; USR 32550
 40 GO TO 10
999 STOP
```

Entrez 65, puis 35, vous aurez 100 évidemment, mais entrez 150, puis 200, et vous aurez 94, c'est-à-dire 350 - 256. Le résultat est modulo 256.

Addition de registres doubles

```
09              ADD HL,BC      LET HL=HL+BC
19              ADD HL,DE      LET HL=HL+DE
```

Voir en table 3, annexe 1, le reste des C.M.

Ces instructions n'influencent que l'indicateur C_i.
On ne peut additionner qu'a HL, IX et IY.

EXERCICE 2 : faîtes un programme en C.M. pour réaliser l'addition modulo 65536. Voici le Basic d'exploitation :

```
10 INPUT "Entrez x",x:POKE 32551,x-INT(x/256)*256: POKE 32552,
   INT (x/256)
20 INPUT "Entrez y",y: POKE 32554,y-INT(y/256)*256: POKE 32555,
   INT (y/256)
30 PRINT x;"+";y;"=";USR 32550
40 GO TO 10
999 STOP
```

Addition tenant compte de C_i

CE,nn	ADC A,N	LET A=A+n+C_i
88	ADC A,B	LET A=A+B+C_i
8E	ADC A,(HL)	LET A=A+PEEK HL+C_i
ED,4A	ADC HL,BC	LET HL=HL+BC+C_i

Avec C_i, on ne peut additionner qu'à A ou HL.

Toutes les instructions ADC influencent les indicateurs S, Z, P/V et C_i.

Il n'y a pas d'instruction pour additionner une constante à HL ; si on veut faire LET HL=HL+64, on entrera :

```
01,40,00        LD BC,64d
09              ADD HL,BC
```

mais, si BC est employé pour autre chose, on fera :

```
7D              LD A,L
06,40           ADD A,64d
6F              LD L,A
7C              LD A,H
CE,00           ADC A,0
67              LD H,A
```

Voyez-vous pourquoi ADC A,0 est nécessaire, ainsi que les deux instructions : précédente et suivante ?

C'est pour additionner le C_i éventuel mis par ADD A,64.

Addition d'une unité ou incrémentation

3C	INC A	LET A=A+1

de même pour les 6 autres registres simples :

```
34              INC (HL)
```

qui réalise l'instruction (HL) = (HL) +1, ou en Basic :

```
LET X=PEEK HL+1  puis  POKE HL,X
```

de même pour (IX+d) et (IY+d).

Les instructions d'incrémentation d'un octet influencent S, Z, P/V, mais *pas* C_i.

```
03              INC BC          LET BC=BC+1
```

de même pour les 5 autres registres doubles.

Les instructions d'incrémentation d'un registre double n'influencent aucun indicateur. Les incrémentations se font, bien sûr, modulo 256. Si A=255 après INC A, A=0.

LES SOUSTRACTIONS

```
D6,nn           SUB A,n         LET A=A-n
90              SUB A,B         LET A=A-B
```

On peut soustraire n'importe quel registre de A.

```
96              SUB A,(HL)      LET A=A-PEEK HL
DD,96,dd        SUB A,(IX+d)    LET A=A-PEEK (IX+d)
FD,96,dd        SUB A,(IY+d)    LET A=A-PEEK (IY+d)
```

Tous les SUB influencent les indicateurs S, Z, P/V et C_i.
On ne peut soustraire que de A. Il n'y a pas de SUB pour les registres doubles.

SUB A,A fait à la fois A=0 et C_i=0, c'est une façon de mettre C_i à 0 quand on n'a pas besoin de A.

SUB A,B veut dire LET A=A-B, mais aussi :

```
IF B > A THEN LET Ci =1
IF B=A   THEN LET Z=1
         etc...
```

Soustraction tenant compte de C_i

```
DE,nn           SBC A,n         LET A=A-n-Ci
98              SBC A,B         LET A=A-B-Ci
```

idem pour les 6 autres registres simples.

```
9E              SBC A,(HL)      LET A=A-PEEK HL -Ci
```

idem pour (IX+d) et IY+d).

```
ED,42           SBC HL,BC       LET HL=HL-BC-Ci
```

idem pour DE, HL et SP.

On ne peut soustraire un octet que de A et deux octets que de HL.

Tous les SBC influencent les 4 indicateurs principaux.

SBC A,A est utile et souvent employé dans la MEM:

si C_i =0 alors A=0 et C_i =0
si C_i =1 alors A=255 et C_i=1

Pour faire SUB HL,BC qui n'existe pas, il faut mettre C_i à 0, puis employer SBC:

```
C6,00          ADD A,0        :Ci =0
ED,42          SBC HL,BC
```

ou bien :

```
97             SUB A,A        :Ci =0
ED,42          SBC HL,BC
```

qui consomme un octet en moins, mais met A à 0.

Pour faire SBC BC,DE qui n'existe pas, on peut employer des instructions d'échange mais, avec nos connaissances actuelles, on fera :

```
60             LD H,B
69             LD L,C
ED,52          SBC HL,DE
44             LD B,H
4D             LD C,L
```

Exemple 5 : soustraction modulo 65536.

```
32550 21,00,00  LD HL,x
32553 01,00,00  LD BC,y
32556 C6,00     ADD A,0        :Ci =0
32558 ED,42     SBC HL,BC
32560 44        LD B,H
32561 4D        LD C,L
32562 C9        RET
```

Employez le Basic de l'exercice 2 en remplaçant, dans la ligne 30 : "+" par "-".

Essayez quelques soustractions pour vous familiariser:

1000 - 1001 = 65535, etc...

Exemple 6 : pour connaître la mémoire libre :

```
32550 21,00,00   LD HL,0
32553 39         ADD HL,SP
32554 ED,4B,
      65,5C      LD BC,(23653)   :(STKEND)
32558 ED,42      SBC HL,BC
32560 44         LD B,H
32561 4D         LD C,L
32562 C9         RET
```

Faîtes PRINT USR 32550 pour avoir le résultat.

Les deux premières lignes exécutent LD HL,SP dont on ne dispose pas.

Pourquoi n'a-t-on pas mis SUB A,A avant SBC pour s'assurer que C_i =0 ? Parce que ADD HL,SP l'a fait.

EXERCICE 3 : faîtes un programme pour connaître la longueur d'un programme Basic en employant les variables système VARS et PROG

Soustraction d'une unité ou décrémentation

```
3D               DEC A          LET A=A-1
```

de même pour les 6 autres registres simples.

```
35               DEC (HL)
```

qui réalise l'instruction (HL) = (HL) - 1

ou, en Basic POKE HL, PEEK HL-1

de même pour (IX+d) et (IY+d).

La décrémentation d'un octet influence S, Z, P/V, mais *pas* C_i

```
0B               DEC BC         LET BC=BC-1
```

de même pour les 5 autres registres doubles.

La décrémentation d'un registre double n'influence aucun indicateur.

Quelques instructions qui influencent la valeur d'un bit

```
3F               CCF            LET Ci=1-Ci
```

c'est le complément restreint de C_i

37	SCF	LET C_i=1

Ces deux instructions ont un effet obligé sur C_i.

CB,87	RES 0,A

réalise l'instruction A0 = 0, donc met le bit 0 de A à 0.

CB,C7	SET 0,A

réalise l'instruction A0 = 1, donc met le bit 0 de A à 1 (voir les 160 C.M. de RES et SET, en annexe 1, table 2).

SET met à 1 et RES met à 0 le bit désigné à gauche de la virgule, d'un registre ou d'un octet mémoire désigné à droite de la virgule.

Chaque cas a une traduction différente en Basic, puisqu'on agit sur un bit. Si on sait que A = 0, le fait de faire SET 7,A exécute LET A=128. D'autre part, si on ne sait pas ce que A contient, faire RES 0,A met le bit 0 de A à 0 = le bit 0 a la valeur 1, donc c'est LET A=A-1, oui, à condition que A soit impair ; si A est pair, alors c'est LET A=A. On peut traduire ceci par :

```
IF INT(A/2) <> A/2 THEN LET A=A-1
```

LES CODES MACHINE REALISANT SIMULTANEMENT PLUSIEURS OPERATIONS DE BASE

Nous avons vu des mnémoniques et leurs C.M. équivalents qui effectuent deux opérations ; en voici 4 qui effectuent 4 opérations et, de ce fait, sont très puissants. Quand vous rencontrez des C.M. semblables, ou leurs mnémoniques, remémorez-vous toutes les opérations qu'ils exécutent, car la suite du programme en tient compte, et cette suite pourrait vous paraître obscure.

ED,A8	LDD	exécute
	(DE) = (HL)	POKE DE,PEEK HL
	DE = DE -1	LET DE=DE-1
	HL = HL -1	LET HL=HL-1
	BC = BC -1	LET BC=BC-1
ED,A0	LDI	exécute
	(DE) = (HL)	
	DE = DE +1	
	HL = HL +1	
	BC = BC -1	

Les trois registres doubles sont utilisés. Ils seront donc chargés à des valeurs convenables, sauf si on utilise ces C.M. pour une instruction particulière. En effet, ces C.M. réalisent (DE)=(HL) qui correspond à LD (nm),(xy) qui n'existe pas. On peut les employer pour cette instruction, quoique LD A,(xy), suivi de LD (nm),A fasse la même chose en employant 6 octets au lieu de 2, mais sans toucher à BC ni à DE.

Ces C.M. agissent sur l'indicateur P/V qui vient à 0 si BC =0 après exécution, sinon P/V =1. C'est le seul DEC BC qui influence un indicateur.

```
ED,B8          LDDR
ED,B0          LDIR
```

exécutent les mêmes opérations que LDD et LDI, mais avec répétition, jusqu'à ce que BC =0. Ces deux C.M. mettent évidemment P/V à 0.

Ils servent à transférer des blocs d'octets d'un endroit à l'autre de la mémoire. Comme BC tient 16 bits, le nombre d'octets concernés peut être grand.

Exemple 7 : supposons que nous ayons introduit 168 octets dans une matrice pour définir 21 nouveaux caractères. A l'aide du programme suivant, nous transférons ces octets dans la zone des graphiques définis par l'utilisateur.

```
32550 97        SUB A,A          :Ci = 0
32551 2A,4B,5C  LD HL,(VARS)     :(23627)
32554 01,06,00  LD BC,6
32557 09        ADD HL,BC
32558 11,58,7F  LD DE,32600      :UDG
32561 01,A8,00  LD BC,168
32564 ED,B0     LDIR
32566 C9        RET
```

Il faut faire la déclaration DIM a$ au début du programme Basic pour que, après exécution, la matrice a$ soit en 1ère position dans la zone variables. Dans cette zone, les 6 premiers octets servent à la gestion de la matrice (voir manuel Sinclair). C'est pour cette raison que nous avons ajouté 6 à l'adresse du début de la zone variables.

Comme illustration, entrez le Basic suivant :

```
 1 DIM a$(168)
10 FOR n=1 TO 168
20 LET a$(n)= CHR$ n
30 NEXT n
40 RANDOMIZE USR 32550
50 FOR n=0 TO 167
60 PRINT PEEK (32600+n);" ";
70 NEXT n
999 STOP
```

Les lignes Basic 10, 20 et 30 chargent la matrice avec des chiffres de 1 à 168. La ligne 40 transfère ces octets dans UDG. Les lignes 50, 60 et 70 affichent ce qu'il y a dans cette zone graphiques utilisateurs pour contrôle.

Exemple 8 : voici un programme en C.M. qui réalise CLS

```
32550 21,FF,57  LD HL,22527
32553 11,DF,57  LD DE,22495     :HL - 32
32556 01,E0,17  LD BC,6112      :6144 - 32
32559 ED,B8     LDDR
32561 C9        RET
```

Le fichier d'affichage est complexe et nous l'utiliserons plus loin. Cependant, pour l'effacer, c'est facile en C.M. On charge HL avec le dernier octet du fichier, DE avec HL moins 32 et BC avec 6112, soit 191 fois 92 ; car on part de la 192e ligne de pixels (qui y sont à 0) qu'on place dans la 191e, puis celle-ci dans la 190e, etc... Puis LDDR.

Faîtes le Basic :

```
 10 LET a$="  32 caractères au choix "
 20 FOR n=0 TO 21
 30 PRINT AT n,0;a$
 40 NEXT n
 50 PAUSE 100
 60 RANDOMIZE USR 32550
999 STOP
```

Les lignes 10 à 40 remplissent l'écran avec un caractère de votre choix. La ligne 50 attend deux secondes. La ligne 60 exécute CLS. Voyez à quelle vitesse l'écran s'efface.

Vous pouvez effacer une partie de l'écran : le dessus, le milieu ou le dessous. Vous devez partir d'une ligne de pixels qui soient tous à 0. C'est, en général, le cas pour les 8e octets des matrices des caractères affichables (à l'exception des minuscules avec jambage). Vous adaptez ensuite HL, DE et BC.

Exemple 9 : manipulons les couleurs. Voici un petit programme qui exploite le fichier des attributs. Chaque position PRINT d'un caractère possède un octet d'attribut. Nous chargeons ce fichier d'attributs avec des octets pris au hazard dans la MEM.

```
32550 21,00,19     LD HL,6400
32553 11,00,58     LD DE,22528
32556 01,00,03     LD BC,768
32559 ED,BO        LDIR
32561 C9           RET
```

Faîtes le Basic :

```
 10 RANDOMIZE USR 32550
 20 PAUSE 200
999 STOP
```

Voyez l'écran se remplir instantanément de couleurs, de clignotement et de sur-brillance au hazard.

Si vous faîtes POKE 32552,61, on transfère les octets de la REM qui représentent les matrices des caractères et, comme ceux-ci ont très souvent leurs 1er et 8e octet à 0, vous verrez prédominer les caractères noirs. Si vous faîtes POKE 32552,0, vous aurez le début de la REM qui contient certaines tables, et la prédominance de certaines couleurs y est aussi visible.

Nous avons mis cette routine à la suite de l'autre (de l'exemple 6) pour que vous puissiez les employer à votre guise. Veuillez noter que celle-ci peut remplacer un DEF FN du Basic alors que celle de l'exemple 6 ne le peut pas puisqu'on y emploie l'adresse de SP qui n'est pas accessible au Basic.

Nous avons mis la ligne 20 pour que, pendant 4 secondes, vous voyiez l'écran rempli entièrement, ce que permettent les C.M. La ligne 999 exécute un STOP, et les deux dernières lignes s'effacent pour laisser place au compte-rendu. Si vous voulez examiner l'écran plus à votre aise, faîtes 20 PAUSE 0 ; il suffira que vous appuyiez sur ENTER pour faire apparaître le compte-rendu. Vous pouvez vous exercer à remplir le fichier d'attributs à partir d'une matrice que vous aurez préalablement remplie.

CORRIGÉS DES EXERCICES

REPONSE AUX EXERCICES

EXERCICE 1

Pour le numéro de colonne :

```
32550 ED,4B,88,5C        LD BC,(23688)
32554 06,00              LD B,0
32556 C9                 RET
```

et pour le numéro de ligne :

```
32557 ED,4B,89,5C        LD BC,(23689)
32561 06,00              LD B,0
32563 C9                 RET
```

PRINT USR 32550 vous donne le numéro de colonne.
PRINT USR 32557 vous donne le numéro de ligne.
PRINT USR 32557,USR 32550 vous donne les deux : ligne, puis colonne.

EXERCICE 2

```
32550 21,00,00           LD HL,x
32553 01,00,00           LD BC,y
32556 09                 ADD HL,BC
32557 44                 LD B,H
32558 4D                 LD C,L
32559 C9                 RET
```

Entrez 4500 puis 560, vous aurez 5060.
Entrez 65535 puis 2, vous aurez 1.

EXERCICE 3

```
32563 ED,5B,53,5C        LD DE,(23635)    :(PROG)
32567 2A,4B,5C           LD HL,(23627)    :(VARS)
32570 97                 SUB A,A
32571 ED,52              SBC HL,DE
32573 44                 LD B,H
32574 4D                 LD C,L
32575 C9                 RET
```

PRINT USR 32563 donne le résultat

CHAPITRE III

BOUCLES TESTS ET OPÉRATIONS COMPLEXES

Le chapitre précédent a familiarisé le lecteur avec les opérations de base du langage machine, celles qui modifient simplement les différents registres du micro-processeur.

Les instructions, sur lesquelles repose la strutura-tion des programmes : boucles, tests et comparaisons, sont maintenant étudiées, ainsi que certaines opérations qui modifient les registres de façon complexe, sur la base de comparaisons logiques, de glissements et de rotations.

LES SAUTS ET LES BOUCLES

Les sauts inconditionnels

C3,qq,pp	JP pq	GO TO pq

réalise l'instruction PC = pq

Pour rappel, l'octet bas, puis l'octet haut de l'adresse sont introduits dans cet ordre. Signalons la seule exception : le numéro d'une ligne d'un programme Basic est codé sur deux octets : l'octet haut, puis l'octet bas.

E9	JP HL	GO TO HL
DD,E9	JP IX	GO TO IX
FD,E9	JP IY	GO TO IY

Ces quatres instructions sont des sauts inconditionnels à une adresse, quelle qu'elle soit.

JL HL est intéressant, il correspond à un GO TO A où A est une variable. HL peut résulter d'un calcul, tout comme A : ce qui produit un GO TO calculé. Attention cependant au calcul de l'expression parce qu'il ne faut pas faire de JP dans le moniteur, le retour ne s'effectuerait pas dans le programme Basic.

```
18,ee            JR e          LET PC=PC+e+2
```

e est un nombre avec signe, tout comme le déplacement d, agissant sur un registre d'index que nous avons décrit au chapitre précédent.

Les valeurs possibles vont de -128 à +127, et les sauts relatifs possibles vont de -126 à +129, en comptant à partir de l'adresse du premier octet de JR. Si on a :

```
32500 18,03      JR 3
```

le saut est compté à partir de 32500 et l'instruction suivante sera celle qui est 32505. Il en résulte que :

- JR 0 n'a pas d'effet ;
- JR FEh est une boucle infinie.

JR agit comme JP, mais dans un espace plus restreint. L'avantage de JR, qui n'emploie que deux octets, est de pouvoir reloger la routine qui l'exploite ; puisque le saut est relatif, aucune adresse absolue n'est donnée.

Tous ces sauts n'ont pas d'effet sur les indicateurs.

Boucle utilisant le registre B comme compteur

```
10,ee            DJNZ e
```

est une instruction puissante et très souvent employée qui réalise plusieurs opérations :

```
    B = B-1
si B ≠ 0, faire PC = PC+e+2
si B = 0, poursuivre avec PC = PC+2
```

e est un nombre avec signe que l'on peut aussi appeler *déplacement*. On ajoute 2, comme avec JR e, pour tenir compte des deux octets de C.M. DJNZ n'a pas d'effet sur les indicateurs, donc DJNZ 0 fait la même chose que DEC B, mais sans influencer les indicateurs. La boucle est relogeable dans son intégralité. Si on ajoute ou retranche une instruction de la boucle, on doit modifier l'argument e en conséquence. La traduction Basic se présente comme suit :

```
          LD B,20h       FOR B=32 TO 0 STEP -1
       ┌─►instruction    commande Basic
       │  instruction    commande Basic
       └──instruction    commande Basic
          DJNZ e         NEXT B
```

B sert de compteur ; à la fin de la boucle, B est à 0.

Par exemple, si on veut ajouter 10 à DE, on fait :

```
LD HL,10
ADD HL,DE
LD D,H
LD E,L
```

ou bien

```
          LD B,10
       ┌─►INC DE
       └──DJNZ FDh
```

sans employer HL.

Exemple 1 - Blanchir une ligne de pixels.

N.B. : en employant le programme de chargement du chapitre 1, entrez 99 comme nombre d'octets pour les exercices de ce chapitre.

```
32500 21,00,40   LD HL,16384  : le ligne
32503 06,20      LD B,32
32505 36,00    ┌─►LD (HL),0
32507 23       │  INC HL
32508 10,FB    └──DJNZ -5+2
32510 C9         RET
```

Vous aurez souvent besoin de cette petite routine pour blanchir ou noircir une ligne de pixels. Nous expliquerons les adresses des pixels dans le chapitre ultérieur mais déjà, à titre d'exemple, noircissons la première ligne de pixels en faisant :

```
    POKE 32506,255   pour réaliser LD (HL),255
ensuite,  CLS        puis RANDOMIZE USR 32500

    POKE 32506,204   suivi de RANDOMIZE USR 32500
```

donnera de petits tirets de deux pixels,

```
    POKE 32506,170   puis RANDOMIZE USR 32500
```

produira des points d'un seul pixel que l'on ne pourra voir qu'avec un bon téléviseur ou un bon moniteur : en effet, une définition de 256 points sur une ligne, qui n'est même pas une ligne de balayage horizontale complète, est de la haute définition. Ne pas oublier que l'affichage se fait sur une partie de l'écran, sur une fenêtre en quelque sorte, pas sur sa totalité.

Les sauts conditionnels

C2,qq,pp	JP NZ,pq	si Z=0
CA,qq,pp	JP Z,pq	si Z=1
D2,qq,pp	JP NC,pq	si C_i=0
DA,qq,pp	JP C,pq	si C_i=1
E2,qq,pp	JP PO,pq	si P/V=0
EA,qq,pp	JP PE,pq	si P/V=1
F2,qq,pp	JP P,pq	si S=0
FA,qq,pp	JP M,pq	si S=1

Ces codes correspondent à l'alternative suivante :

- si la condition est vraie, faire PC=pq ;
- si la condition est fausse, poursuivre avec PC=PC+3.

Les codes ci-dessous :

20,ee	JR NZ,e	si Z=0
28,ee	JR Z,e	si Z=1
30,ee	JR NC,e	si C_i=0
38,ee	JR C,e	si C_i=1

équivalent au jeu de conditions suivant :

- si la condition est vraie, faire PC=PC+e+2 ;
- si la condition est fausse, poursuivre avec PC=PC+2.

Les sauts conditionnels n'ont pas d'effet sur les indicateurs. Les sauts conditionnels relatifs sont relogeables, ils testent moins d'indicateurs et effectuent des sauts dans un espace plus restreint que les sauts conditionnels simples. Aussi, sont-ils souvent employés.

Les premiers sont nécessaires quand on teste P/V ou S, ou quand le saut doit être plus grand que 129 ou supérieur à -126 pour les retours en arrière.

Les sauts conditionnels permettent les IF... THEN... et FOR... NEXT... voulus. Ils s'emploient après une instruction qui modifie l'indicateur concerné.

IF C_i THEN GO TO...
IF NOT C_i THEN GO TO...
IF Z THEN GO TO...
etc.

```
SUB A,B
←JP Z,pq
```

signifient IF A-B=0 THEN GO TO pq.

Boucles imbriquées

Nous avons vu que DJNZ exécute une boucle avec décrémentation automatique de B. Si on utilise plusieurs boucles imbriquées, on emploie alors un ou plusieurs autres registres, selon le même principe.

Ainsi :

```
        DEC C
←       JR NZ,e
```

équivaut à un DJNZ e avec C comme compteur.

La souplesse d'utilisation est plus grande, comme dans cet exemple :

```
        LD C,236        FOR C=0 TO 20
┌──→    instruction     commande Basic
│       instruction     commande Basic
│       instruction     commande Basic
│       INC C          }
└──     JR NZ,e       } NEXT C
```

où la boucle est exécutée par incrémentation de C. Quand C a été incrémenté 20 fois, C=0 d'où Z=1 et on quitte la boucle.

```
        LD C,10         FOR C=10 TO 0
┌──→    instruction     commande Basic boucle C
│       instruction     commande Basic boucle C
│       LD D,156        FOR D=0 TO 100
│  ┌→   instruction     commande Basic boucle D
│  │    instruction     commande Basic boucle D
│  │    INC D          STEP +1
│  └──  JR NZ,e
│       instruction     commande Basic boucle C
│       DEC C          STEP -1
└──     JR NZ,e
```

La boucle C sera exécutée 10 fois et, pour chaque C, la boucle D sera exécutée 100 fois.

Remarquons, au passage, un mnémonique intéressant :

```
00              NOP
```

qui n'effectue aucune opération. NOP sert à introduire de courts délais, 4 cycles d'horloge, soit 1,14 microsec. et à corriger les programmes au cours de leurs mises au point. Il est rare qu'un programme tourne correctement du premier jet : il faut donc modifier quelques instructions et, si vous avez glissé quelques NOP dans votre programme, ils seront utiles pour ajouter quelques codes

supplémentaires sans devoir recalculer toutes les adresses. Inversement, quand on supprime des instructions, il est utile de les remplacer par des NOP.

Exemple 2 - Faire défiler l'écran d'un caractère vers la gauche, en rotation complète.

```
32500 00,00        DEF            : stocke BC
32502 21,01,40     LD HL,16385
32505 11,00,40     LD DE,16384
32508 01,C0,00     LD BC,192
32511 ED,43,F4,    LD (32500),BC
          7E
32515 01,1F,00     LD BC,31
32518 1A           LD A,(DE)
32519 ED,B0        LDIR
32521 00           NOP
32522 00           NOP
32523 12           LD (DE),A
32524 13           INC DE
32525 23           INC HL
32526 ED,4B,F4,    LD BC,(32500)
          7E
32530 OD           DEC C
32531 20,EA        JR NZ, -22+2
32533 C9           RET
```

Nous avons réservé les deux premiers octets. Le mnémonique DEF ne signifie rien : il signifie *déf*inition ; nous avions besoin d'un registre double supplémentaire pour stocker BC provisoirement.

LDIR s'emploie pour faire défiler tous les octets d'une ligne de pixel vers la gauche. On construit une boucle pour effectuer cette opération 192 fois avec C, comme compteur ; au début de la boucle, BC est stocké dans 32500 et 32501 et, à la fin de la boucle, BC est récupéré. DEC C est nécessaire au lieu de DEC BC, parce que DEC BC n'influence pas l'indicateur Z. LD A,(DE) stocke l'octet de la 1ère colonne de gauche pour le remettre après LDIR dans la dernière colonne de droite. Deux NOP sont prévus pour une opération supplémentaire.

Pour exploiter la routine ci-dessus, entrer le programme Basic suivant :

```
10 FOR n=1 TO 22
20 PRINT "0123456789abcdefghijklmnopqrstuv"
30 NEXT n
40 FOR n=1 TO 32
50 RANDOMIZE USR 32502
60 PAUSE 50
```

```
 70 NEXT n
999 STOP      ... RUN
```

PAUSE 50 vous permet de voir à quelle vitesse toutes les colonnes se déplacent vers la gauche. Supprimez la ligne 60 et le défilement est plus rapide. Au lieu de prendre la 1ère colonne pour la remettre dans la 32e, on peut blanchir à partir de la 32e en faisant LD A,0 à la place des NOP. Essayer : POKE 32521,62, puis RUN.

EXERCICE 1 : *Ecrivez un programme pour faire défiler, en rotation complète, les colonnes de caractères vers la droite, sur tout l'écran.*

LES CODES MACHINE REALISANT LES OPERATIONS LOGIQUES

Les opérations logiques, par le microprocesseur, sont comparables, dans leur principe, aux fonctions logiques du Basic, mais pas dans leur fonctionnement : elles agissent, en effet, sur des registres par calcul bit à bit.

L'opération ET

E6,nn AND n

réalise l'opération A=A ∧ n

A0 AND B

exécute A=A∧B. Il en va de même pour les 7 registres simples.

A6 AND (HL)

effectue A=A∧ (HL) ; de même pour IX+d et IY+d.

On ne peut réaliser l'opération ET qu'avec le registre A. Cette opération agit sur les indicateurs S, Z, P/V et met C_i à 0.

Si on souhaite obtenir B=B ∧ C, pour lequel n'existe pas de code spécialisé, il faut procéder ainsi :

```
LD A,B
AND C
LD B,A
```

AND A laisse A inchangé et met C_i =0. Souvent employé pour cette raison, AND O met A à 0 ; c'est la même chose

que LD A,0, excepté que cette dernière instruction n'influence pas les indicateurs. AND FFh laisse A inchangé.

Le ET logique sert souvent à masquer certains bits pour les faire disparaître. Ceci nous sera utile pour les différents calculs que le fichier d'affichage nécessite et pour réaliser des modulo 8 pour manipuler les attributs.

AND 07 après l'opération, A aura une valeur comprise entre 0 et 7 : c'est donc A modulo 8.
AND 06, par contre, donnera à A les valeurs 0, 2, 4 ou 6.
AND C0 laisse les bits 7 et 6 inchangés, les autres vont à 0.
AND 0F laisse le quartet bas inchangé, l'autre va à 0.

Nous vous conseillons, pour tous ces exemples, de reprendre les règles du chapitre I et de les appliquer pour chaque cas.

L'opération OU

F6,nn OR n

équivaut à l'opération : A = A V n

B0 OR B

est identique à : A = A V B ; cette opération est possible avec les 7 registres simples.

B6 OR (HL)

pour les octets en mémoire ou pour IX+d et IY+d.

L'opération OU ne se fait qu'avec A et agit sur les indicateurs S, Z, P/V et C_i en le mettant à 0.

On emploie LD pour réaliser des OU logiques avec un autre registre que A.

Si A=0, alors OR n est le même que LD A,n, sauf pour les indicateurs ; si A=FF, alors OR n laisse A inchangé. On se sert souvent de OR pour mettre certains bits à 1, tout en laissant les autres inchangés. OR C0h met les bits 7 et 6 de A à 1, et les autres restent inchangés. Si A contenait les attributs d'un caractère, ceci met le clignotement et la brillance sans changer la couleur de l'encre ni du fond.

L'opération OU exclusif

```
     EE,nn              XOR n
```

réalise l'opération A = A ⊻ n

```
     A8                 XOR B
```

A = A ⊻ B ; il en va de même avec les 7 registres simples.

```
     AE                 XOR (HL)
```

(octet mémoire IX+d et IY+D, HL).

Les OU exclusifs ne se font qu'avec A et influencent S, Z, P/V et C_i qui vient à 0.

La manipulation de XOR est difficile. Appliquez les règles du chapitre I pour connaître le résultat.

XOR, appliqué deux fois de suite, retourne le A original. Ce fait a reçu une application avec le OVER du Basic : cet OVER se traduit en C.M., comme nous le montrons dans un exemple ci-après.

XOR A est utile pour faire à la fois A=0 et C_i=0. A noter que cette instruction met aussi S à 0, Z à 1 et P/V à 1 (indicateur P), mais on l'emploie souvent pour ce qui précède immédiatement. Dans l'exemple 2 précédent, on peut remplacer le 1er NOP par XOR A, on aura le même résultat que LD A,0, en faisant la commande POKE 32521, 175.

Le complément

```
     2F                 CPL          LET A=1-A
```

exécute A = $\overline{A}$, c'est le complément restreint de A. Pas d'effet sur les indicateurs. Si on veut obtenir le complément de BC :

```
     78                 LD A,B
     2F                 CPL
     47                 LD B,A
     79                 LD A,C
     2F                 CPL
     4F                 LD C,A
```

et, si on veut le complément vrai, on ajoute :

```
     03                 INC BC
          ED,44              NEG       LET A=0-A
```

exécute le complément vrai de A.

NEG agit sur les indicateurs :

- S et Z sont mis suivant le résultat.
- C_i = 1, si A contenait 0 avant l'opération NEG.
- P/V = 1 si A contenait 80h avant l'opération NEG.

Le complément vrai de 80h est 80h.

- si A = 1, NEG fait A = -1 ;
- si A =-6, NEG fait A = 6.

CPL, suivi de INC A, a le même effet que NEG, mais sans influencer C_i, les autres indicateurs changent avec INC A.

Voici encore un C.M. qui nous sera très utile plus loin.

```
76              HALT
```

Cette instruction exécute des NOP jusqu'à une interruption ou un reset du microprocesseur. Les circuits du ZX-Spectrum forcent une interruption du type $\overline{INT}$ à chaque affichage, tous les 1/50ème de sec. Par conséquent, HALT servira à synchroniser notre programme avec la production d'une image.

Exemple 3 - Réaliser des surimpressions comme le Basic OVER.

```
32500 00,00       DEF           : stocke BC
32502 21,00,40    LD HL,16384   : à surimprimer
32505 11,00,48    LD DE,18432   : qui surimprime
32508 01,00,08    LD BC,2048    : nombre d'octets
32511 ED,43,
      F4,7E     ┌─►LD (32500),BC
32515 46        │  LD B,(HL)
32516 1A        │  LD A,(DE)
32517 A8        │  XOR B
32518 77        │  LD (HL),A
32519 ED,4B,    │
      F4,7E     │  LD BC,(32500)
32523 23        │  INC HL
32524 13        │  INC DE
32525 0B        │  DEC BC
32526 78        │  LD A,B
32527 B1        │  OR C
32528 20,EA     └──JR NZ,-19+2
32530 C9          RET
```

L'emploi de DEF a déjà été vu. La boucle est contrôlée par le compteur BC car elle est effectuée 2048 fois, il nous fallait donc un registre double. La partie intéres-

sante suit DEC BC qui n'influence pas les indicateurs, alors que LD A,B, suivi de OR C, réalise B OR C= quand tous les bits de B et de C seront à 0, l'indicateur Z viendra à 1.

XOR est la fonction employée pour réaliser le OVER du Basic, mais il est évident qu'une autre fonction peut être programmée.

Ce programme surimprime les 8 premières lignes avec les lignes 9 à 16. En changeant les chargements de HL, DE et BC au départ, il est possible de réaliser d'autres surimpressions, mais il est préférable d'attendre, pour cela, les explications qui seront données dans les chapitres ultérieurs.

A titre d'illustration, laissez l'écran tel quel après le chargement de ce programme. L'affichage vous montre les lignes 32511 à 32530. Faîtes la commande RANDOMIZE USR 32502.

Si vous faîtes une 2ième fois cette commande, vous retrouvez l'affichage initial : c'est bien l'action de la commande OVER ; ainsi, les chiffres 325 surimprimés par des 325 disparaissent, etc.

LES TESTS ET COMPARAISONS

Les tests de chaque bit pris isolément

```
CB,47          BIT 0,A      LET Z=1-A0
```

exécute l'instruction Z=A0 et agit, par conséquent, sur l'indicateur Z. C'est le complément de 1 bit (voir les 80 C.M. des mnémoniques BIT en annexe).

BIT donne le moyen de tester n'importe quel bit de n'importe quel registre ou de n'importe quel octet en mémoire. Les sauts conditionnels sur Z ou NZ s'emploient ensuite pour décider l'instruction suivante.

Les codes des caractères affichables vont de 20 à 7Fh. Des tests, sur les bits du quartet haut, permettent leur identification. Par exemple, les lettres majuscules vont de 41h à 5Ah, et les minuscules de 61h à 7Ah. Les minuscules ont les mêmes codes que les majuscules, mais avec le bit 5 à 1 ou, si vous préférez, ont 32d de plus.

Les comparaisons

```
FE,nn          CP n          exécute A-n
          ou
B8             CP B           xécute A-B
          ou
BE             CP (HL)       exécute A-(HL)
```

Les comparaisons d'une constante, d'un registre ou d'un octet mémoire ne se font qu'avec A. Voir le reste des C.M. en annexe.

Les mnémoniques CP effectuent une soustraction, mais sans modifier d'octets ; il n'y a pas de résultat. Les indicateurs sont influencés par cette opération, comme si elle avait lieu. Ces instructions sont très souvent employées, suivies d'un saut conditionnel. La correspondance Basic est :

IF comparaison THEN saut ou décision.

Par exemple, CP 3 fait :

```
IF A > = 3 THEN LET Ci=0
IF A < 3 THEN LET Ci=1
```

mais aussi :

```
IF A=3 THEN LET Z=1
IF A<>3 THEN LET Z=0
etc.
```

Les trois codes ci-dessous permettent de savoir si un registre est à 0 :

```
LD A,0
CP C              IF C=0
JR Z,e            THEN GO TO
```

Le THEN sera toujours un THEN GO TO, mais, à l'adresse de transfert, vous mettez les équivalents de PRINT, LET, etc., selon le but recherché. Vous pouvez également transposer une nouvelle commande Basic telle que IF... THEN... ELSE... en écrivant, par exemple, IF C=0 THEN LET C=31 ELSE LET C=C+1 :

```
LD A,0
CP C
JR Z,+3+2
INC C
JR 2+2
LD C,1Fh
suite des instructions
```

Exemple 4 - Inverser les pixels d'un fichier et, de cette façon, réaliser la fonction INVERSE.

```
32500 21,00,40   LD HL,16384
32503 7E      ┌─>LD A,(HL)
32504 2F      │  CPL
32505 77      │  LD (HL),A
32506 23      │  INC HL
32507 7C      │  LD A,H
32508 FE,58   │  CP 88d        : 22528
32510 20,F7   └──JR NZ,-9+2
32512 C9         RET
```

L'exemple 3 montrait la fonction OVER en employant deux fichiers. Ici, on met en oeuvre le mnémonique CPL sur un fichier qui correspond à la fonction INVERSE. L'emploi de H pour contrôler la boucle sera expliqué au chapitre suivant.

Exemple 5 - Echanger deux fichiers.

```
32500 21,00,40   LD HL,16384
32503 11,00,48   LD DE,18432
32506 1A         LD A,(DE)
32507 46         LD B,(HL)
32508 77         LD (HL),A
32509 78         LD A,B
32510 12         LD (DE),A
32511 13         INC DE
32512 23         INC HL
32513 7C         LD A,H
32514 FE,48      CP 72d        : 18432
32516 20,F4      JR NZ,-12+2
32518 C9         RET
```

Cette routine est intéressante pour échanger deux fichiers entre eux. Les applications en sont multiples. Ceux qui possèdent un ZX-Spectrum avec 48 K ont la possibilité de stocker 5 images ; cette routine fera apparaître instantanément n'importe quelle image, sans perte de la précédente.

Cette routine se comprend d'elle-même. Ici, elle est construite pour échanger les lignes 1 à 8 avec les lignes 9 à 16 de l'affichage. RANDOMIZE USR 32500 matérialise cet échange de 2048 octets.

Exemple 6 - Pour contrôler le clignotement.

```
32500 00         DEF           : stocke le clignote-
                                 ment
32501 00         DEF           : réserve
32502 21,00,58   LD HL,22528
32505 3A,F4,7E   LD A,(32500)
```

```
32508 CB,7F       BIT 7,A       : Z = A7
32510 28,10     ┌─JR Z,+16+2
32512 3E,FF     │ LD A,FF       : pour SET
32514 32,07,7F  │ LD (32519),A  : se modifie
32517 7E        │►LD A,(HL)
32518 CB,FF     │ SET 7,A       : ou CB,BF pour RES
32510 77        │ LD (HL),A
32521 23        │ INC HL
32522 7C        │ LD A,H
32523 FE,5B     │ CP 91d
32525 20,F6     │►JR NZ,-10+2
32527 C9        │ RET
32528 3E,BF     └►LD A,BF       : pour RES
32530 32,07,7F    LD (32519),A
32533 18,EE       JR -18+2
```

Il n'y a que deux remarques importantes concernant ce programme : quoiqu'il ne contienne que des sauts relatifs JR, il n'est pas relogeable, parce qu'il y a deux instructions qui chargent à une adresse absolue : 32519. Cette adresse devra être changée si vous implantez cette routine à un autre endroit. En 2ème lieu, ceci est un exemple d'un programme qui se modifie lui-même : l'octet 32519 peut contenir FF ou BF. L'instruction se présente comme CB,FF, soit SET 7,A, ou comme CB,BF, soit RES 7,A.

Pour contrôler ce programme, après avoir entré "s" pour arrêter le programme de chargement, faîtes POKE 32500,128, puis RANDOMIZE USR 32502.

EXERCICE 2 - *Ecrire un programme pour contrôler la brillance. Pour charger à partir de 32535 sans devoir faire une série de l pour lister les C.M. jusqu'à 32534. faîtes la commande GO TO 1090, puis entrez vos codes. Il est évident que ceci n'est valable qu'à la condition que le programme Basic éventuel, que vous auriez introduit entre 1 et 999, ne contienne pas les variables a ni a$.*

Les comparaisons avec répétition

```
ED,A9              CPD
```

réalise plusieurs instructions.

```
 A - (HL)
HL = HL -1
BC = BC -1
```

qu'on peut traduire par IF A=PEEK HL THEN...

```
ED,A1              CPI
```

équivaut à :

```
 A - (HL)
HL = HL +1
BC = BC -1

ED,B9              CPDR
ED,B1              CPIR
```

effectue les mêmes opérations que CPD et CPI, mais avec répétition jusqu'à ce que A = (HL) ou que BC = 0.

Ces mnémoniques sont très puissants et très utiles pour retrouver la 1ère occurence d'un caractère dans un fichier ou pour rechercher un bloc.

Ils agissent sur S, *pas sur* C_i, sur Z comme suit :

Z = 1 si A = (HL)

et sur P/V comme suit :

P/V = 0 si BC = 0 après exécution, sinon P/V = 1.

Après ces instructions, le programme se poursuit tout simplement ou un saut conditionnel est effectué selon le but recherché.

Les instructions CP, avec répétition, permettent des milliers d'instructions avec un seul mnémonique.

Exemple 7 - Renumérotation.

Beaucoup de programmeurs aiment renuméroter leurs programmes Basic ; c'est à leur intention que nous donnons cette routine qui, d'autre part, illustre les mnémoniques CPIR et CPI.

```
32500 00,00       DEF             : pour stocker HL
32502 2A,53,5C    LD HL,(23635): (PROG)
32505 11,00,00    LD DE,0
32508 06,0A    ┌─►LD B,10d
32510 13       │  INC DE
32511 10,FD    │  DJNZ -3+2
32513 72       │  LD (HL),D
32514 23       │  INC HL
32515 73       │  LD (HL),E
32516 23       │  INC HL
32517 23       │  INC HL
32518 23       │  INC HL
32519 3E,0D    │  LD A,13d       : caractère de
               │                   contrôle ENTER
32521 0E,00    └─►LD C,0         : pour éviter BC = 0
```

```
32523 ED,B1    ↑  CPIR
32525 ED,4B,
      4B,5C       LD BC,(23627) : (VARS)
32529 22,F4,7E    LD (32500),HL : stocke HL
32532 A7          AND A         : Ci = 0
32533 ED,42       SBC HL,BC
32535 2A,F4,7E    LD HL,(32500): restaure HL
32538 38,EO       JR C,-32+2
32540 00          NOP
32541 2A,53,5C    LD HL,(23635) : (PROG)
32544 3E,EC       LD A,236      : code GO TO
32546 BE          CP (HL)
32547 28,01       JR Z, +1+2
32549 3C          INC A         : code GOSUB
32550 ED,A1       CPI
32552 28,10       JR Z, +16+2
32554 22,F4,7E    LD (32500),HL : stocke HL
32557 ED,4B,
      4B,5C       LD BC,(23627): (VARS)
32561 A7          AND A         : Ci=0
32562 ED,42       SBC HL,BC
32564 2A,F4,7E    LD HL,(32500)
32567 38,E7       JR C, -25+2
32569 C9          RET
32570 3E,0E       LD A,14       : caractère de
                                  contrôle NUMBER
32572 BE          CP (HL)
32573 28,E1       JR Z, -31+2
32575 3E,8F       LD A,143      : caractère pavé noir
32577 77          LD (HL),A
32578 23          INC HL
32579 18,F5       JR -11+2
```

Le nouveau numéro de ligne Basic est dans DE, ligne 32505, et l'incrémentation dans la ligne 32508. Changez les octets 32506, 32507 et 32509, si vous désirez d'autres numéros de ligne et une autre incrémentation.

Avant CPIR, on met C à 0 ; B y est déjà à la suite de la boucle DJNZ. Donc, BC = 0 et la première décrémentation amène BC = 65535 ; on est certain que CPIR s'arrêtera pour A = (HL). Le caractère de contrôle de code 13, que l'on charge dans A, termine chaque ligne Basic. A ce moment, on renumérote à nouveau, sauf si on atteint la zone variables. Cette routine s'arrête à 32540 où vous pouvez entrer C9 (201d).

Si vous laissez NOP, le programme se poursuit avec une autre routine qui a pour but d'attirer votre attention sur les GO TO et GOSUB dont vous devrez changer les numéros de ligne en y substituant des pavés noirs. Le

CP (HL) compare à GO TO et CPI compare à GOSUB et incrémente HL. La boucle de comparaison recommence, sauf si on a atteint la zone variables. D'autre part, si on rencontre GO TO ou GOSUB, on saute à 32570 où on compare à 14 qui est le code du contrôle NUMBER et qui signifie que les 5 octets suivants représentent un nombre en virgule flottante, et donc que les caractères qui précèdent représentent le nombre dont on remplace chaque chiffre par un pavé noir.

LES ROTATIONS ET GLISSEMENTS

Les rotations

17 RLA

exécute une rotation de 9 bits à gauche avec l'indicateur C_i.

07 RLCA

exécute une rotation de 8 bits à gauche. Le bit 7 de A va dans le bit 0 de A et dans C_i.

1F RRA

exécute une rotation de 9 bits à droite avec C_i.

OF RRCA

exécute une rotation de 8 bits à droite. Le bit 0 va dans le bit 7 et dans C_i.

Ces 4 instructions, qui agissent sur A, n'influencent que l'indicateur C_i.

Par contre, les rotations d'un registre ou d'un octet mémoire, dont vous trouverez les C.M. en annexe, exécutent les mêmes opérations, mais agissent sur les 4 indicateurs principaux. D onc :

```
     17          RLA
et   CB,17       RL A
```

font la même chose, excepté sur les indicateurs.

Remarque : Notez que l'espace, qui suit les premières majuscules du mnémonique, a de l'importance. RLA n'est pas RL A ; CPL n'est pas CP L, etc.

Les glissements

CB,3F SRL A

effectue un glissement logique à droite. 0 est mis dans le bit 7 et le bit 0 va dans C_i.

CB,2F SRA A

effectue un glissement arithmétique à droite. Le bit 7 ne change pas et le bit 0 va dans C_i.

CB,27 SLA A

effectue un glissement arithmétique à gauche. 0 est mis dans le bit 0 et le bit 7 va dans C_i.

Les glissements agissent sur les 4 incicateurs. Les glissements d'un registre ou d'un octet en mémoire ont leurs C.M. en annexe.

Après un SRA, le bit 6 sera toujours le même que le bit 7. L'effet de SRA est de diviser par 2 les nombres positifs et négatifs (avec signe). FC (qui représente -4) devient FE (qui représente -2), *mais attention aux nombres impairs* : FD (-3) devient FE (-2), mais avec un C_i=1. On fera donc un test sur C_i ou on admettra que le résultat soit un entier : fonction INT du Basic.

SRL divise par 2 les nombres positifs uniquement.

Exemple 8 - Faire défiler, d'une colonne de pixels, vers la gauche.

```
32500 11,40,00   LD DE,64      : incrément de HL
32503 21,1F,40   LD HL,16415   : début à droite
                                 au dessus
32506 0E,C0      LD C,192      : défile les 192 lignes
32508 06,20    ->LD B,32       : de 32 octets
32510 A7         AND A         : Ci=0 pour mettre 0
                                 dans la colonne de
32511 7E       ->LD A,(HL)     droite
32512 17         RLA
32513 77         LD (HL),A
32514 2B         DEC HL
32515 10,FA      DJNZ -6+2
32517 19         ADD HL,DE
32518 0D         DEC C
32519 20,F3      JR NZ, -13+2
32521 C9         RET
```

On part du point supérieur droit de l'écran. La boucle, contrôlée par B avec DJNZ, comprend RLA qui stocke le bit 7 d'un octet pour le réintroduire dans le bit 0 de

l'octet précédent, et ainsi de suite 32 fois. La boucle, contrôlée par C, permet le défilement des 192 lignes de pixels de l'écran.

Nous donnons un programme Basic d'exploitation, en réponse à l'exercice suivant.

EXERCICE 3 : *Ecrire un programme qui effectue le défilement d'une colonne de pixels vers la droite. Continuez à 32522.*

UTILISATION DE LA PILE ET DES ECHANGES

Les échanges

Nous avons vu, dans le chapitre I, que le microprocesseur possède un 2ème jeu de registre. Voici les C.M. qui permettent d'y accéder.

```
08              EX AF,A'F'
```

réalise deux échanges.

A <=> A' et F <=> F'

F change, donc tous les indicateurs sont modifiés.

En Basic, on aurait pour A :

```
LET X=A
LET A=A'
LET A'=X
```

et de même pour F ;

```
D9              EXX
```

réalise trois échanges des registres couplés.

BC <=> B'C', DE <=> D'E' et HL <=> H'L'

sans effet sur les indicateurs.

```
EB              EX DE,HL
```

provoque l'échange DE <=> HL

```
E3              EX (SP),HL
DD,E3           EX (SP),IX
FD,E3           EX (SP),IY
```

éxécutent chacun deux échanges.

(SP) <=> RRbas et (SP+1) <=> RRhaut,

sans effet sur les indicateurs.

On utilise fréquemment EXX pour sauvegarder provisoirement les 6 registres de travail, afin d'effectuer une routine qui nécessite des registres de travail. Après exécution, on restaure les 6 registres par un 2ième EXX.

```
EXX
routine
EXX
```

Pour faire ADD DE,BC, pour lequel il n'existe pas de code intégré, on emploie :

```
EB              EX DE,HL
09              ADD HL,BC
EB              EX DE,HL
```

EX (SP),HL est intéressant : on pousse HL dans la pile, deux octets de la pile dans HL, et SP ne change pas. C'est un moyen de préserver HL tout en exploitant les deux derniers octets entrés dans la pile.

La pile

La pile est un emplacement de la mémoire qui peut être utilisé pour stocker provisoirement le contenu des principaux registres ; son emploi requiert certaines précautions.

```
C5              PUSH BC
```

exécute plusieurs instructions.

```
(SP-1) = BChaut   (POKE SP-1,B)
(SP-2) = BCbas    (POKE SP-2,C)
SP = SP -2        (LET SP = SP -2)
```

Ceci est un moyen de stocker BC dans la pile. Pour le restaurer, on fait :

```
C1              POP BC
```

qui équivaut aux instructions suivantes :

```
BCbas  = (SP)
BChaut = (SP+1)
SP     =  SP+2
```

Voir, en annexe, les autres C.M. pour les registres doubles et couplés. PUSH et POP n'ont pas d'influence sur les indicateurs. Ces deux instructions sont très employées. Outre le stockage provisoire de registre, elles servent à faire des échanges, modifier une adresse de retour, etc. *Attention à compenser des PUSH par des POP, ne fut-ce que pour désencombrer la pile.* Il y a des exceptions que nous verrons au chapitre suivant.

L'utilisation rationnelle de la pile permet de pallier l'absence de code intégré pour certaines opérations très courantes :

```
LD BC,HL :
      PUSH HL
      POP BC

ADD BC,DE :
      PUSH BC
      POP HL
      ADD HL,DE
      PUSH HL
      POP BC

ADD B,n :
      PUSH AF
      LD A,B
      ADD A,n
      LD B,A
      POP AF
```

La pile permet également la manipulation du registre F.

```
      PUSH AF
      POP BC
```

et le registre F se retrouve dans C ; soit LD C,F.

Soit à exécuter EX D,E, qui n'existe pas, dans modifier aucun registre ; supposons que SP = 30000 :

```
      PUSH DE     (29999) = D
                  (29998) = E
                  SP = 29998

      PUSH DE     (29997) = D
                  (29996) = E
                  SP = 29996

      INC SP      SP = 29997

      POP DE      E  = (29997)
                  D  = (29998)
                  SP = 29999

      INC SP      SP = 30000
```

ou bien, on fera :

```
      PUSH AF
      LD A,D
      LD D,E
      LD E,A
      POP AF
```

CORRIGÉS DES EXERCICES

EXERCICE 1

```
32534 21,FE,57   LD HL,22526
32537 11,FF,57   LD DE,22527
32540 01,C0,00   LD BC,192
32543 ED,43,
      F4,7E      LD (32500),BC
32547 01,1F,00   LD BC,31
32550 1A         LD A,(DE)
32551 ED,B8      LDDR
32553 00         NOP
32554 00         NOP
32555 12         LD (DE),A
32556 1B         DEC DE
32557 2B         DEC HL
32558 ED,4B,
      F4, 7E     LD BC,(32500)
32562 0D         DEC C
32563 20,EA      JR NZ, -22+2
32565 C9         RET
```

C'est le même principe que l'exemple 2, mais on part du coin inférieur droit en employant LDDR.

Au Basic de l'exemple 2, ajoutez :

```
 80 FOR n=1 TO 32
 90 RANDOMIZE USR 32534
100 NEXT n
```

et voyez les deux défilements successifs.

A la place des lignes 10, 20 et 30, vous pouvez afficher autre chose : des cercles, des boites, des signes (au hasard, pour simuler un vol d'oiseaux, etc.).

Vous pouvez ajouter :

```
110 GO TO 40
```

EXERCICE 2

```
32535 21,00,58   LD HL,22528
32538 3A,F4,7E   LD A,(32500)
32541 CB,77      BIT 6,A       : Z = A6
32543 28,10      JR Z, +16+2
32545 3E,F7      LD A,F7       : pour SET
32547 32,28,7F   LD (32552),A  : se modifie
32550 7E         LD A,(HL)
32551 CB,F7      SET 6,A       : ou RES
32553 77         LD (HL),A
32554 23         INC HL
32555 7C         LD A,H
32556 FE,5B      CP 91d
32558 20,F6      JR NZ, -10+2
32560 C9         RET
32561 3E,B7      LD A,B7       : pour RES
32563 32,28,7F   LD (32552),A  : se modifie
32566 18,EE      JR -18+2

      POKE 32500,64
puis  RANDOMIZE USR 32535
```

EXERCICE 3

```
32522 21,00,40   LD HL,16384   : en haut à gauche
32525 0E,C0      LD C,192
32527 06,20      LD B,32
32529 A7         AND A         : Ci=0
32530 7E         LD A,(HL)
32531 1F         RRA
32532 77         LD (HL),A
32533 23         INC HL
32534 10,FA      DJNZ -6+2
32536 0D         DEC C
32537 20,F4      JR NZ, -12+2
32539 C9         RET
```

Même construction que l'exemple 8, mais avec RRA et INC HL. Essayez, par exemple, le Basic suivant :

```
10 PLOT 57,27 : DRAW 120,120,PI*109
20 FOR n=1 TO 30
30 RANDOMIZE USR 32500
40 NEXT n
50 n=1 TO 60
60 RANDOMIZE USR 32522
70 NEXT n
999 STOP
```

CHAPITRE IV

TECHNIQUES SIMPLES APPLIQUÉES A LA COULEUR ET AU SON

Ce chapitre est consacré à de courtes applications C.M. sur deux caractéristiques majeures du ZX-Spectrum : la *couleur* et le *son*. Les techniques simples, qui y sont développées, demontreront, au lecteur, que des effets très intéressants peuvent être aisément obtenus en mettant en oeuvre un jeu de codes relativement réduit.

LA COULEUR

On sait que, sur le ZX-Spectrum, la couleur est définie, pour chaque position d'affichage, par un octet d'attributs où sont codées les différentes caractéristiques de couleur. Il importe donc, pour exploiter les programmes de gestion de couleur et de rotation d'attributs, de bien connaître la position occupée dans la mémoire par les octets concernés.

La gestion des couleurs

Le programme ci-dessous, construit à l'aide de codes simples, a une vocation générale pour la gestion des couleurs (encre et papier) pour chaque position d'affichage.

```
32500 00          DEF             : couleur papier
32501 00          DEF             : couleur encre
32502 21,00,58    LD HL,22528
32505 3A,F4,7E    LD A,(32500)    : couleur papier
32508 E6,07       AND 7           : isole A0, A1 et A2
```

```
32510 87          ADD A,A       : 2A
32511 87          ADD A,A       : 4A
32512 87          ADD A,A       : 8A
32513 47          LD B,A
32514 7E       ┌─►LD A,(HL)
32515 E6,C7    │  AND C7h       : masque la couleur
               │                  précédente
32517 B0       │  OR B          : met la couleur choisie
32518 77       │  LD (HL),A     : l'affiche
32519 23       │  INC HL
32520 3E,5B    │  LD A,91d
32522 94       │  SUB A,H
32523 20,F5    └─ JR NZ -11+2
32525 C9          RET
```

Charger 32500 avec le code couleur choisi, de 0 (noir) à 7 (blanc) pour le papier.

AND 7 élimine les 5 bits hauts, pour le cas où vous auriez entré un nombre plus grand que 7 dans 32500. Les trois ADD multiplient A par 8 et ont pour but d'amener la couleur papier dans les bits A5, A4 et A3.

ANC C7, pour masquer la couleur précédente, et OR B, pour introduire la nouvelle.

Nous vous montrons, ensuite, comment contrôler une boucle sans compteur, ou plutôt avec H, comme compteur en double emploi.

Quand le fichier d'attributs, qui compte 768 octets, est rempli, HL a pour valeur 23296, soit H=91 et L=0, d'où SUB A,H mettra Z à 1 quand H=91.

Comme illustration, faîtes le programme Basic :

```
 10 FOR n=1 TO 22
 20 PRINT "32 caractères"
 30 NEXT n
 40 FOR n=0 TO 7
 50 POKE 32500,n
 60 RANDOMIZE USR 32502
 70 PAUSE 50
 80 NEXT n
999 STOP
```

EXERCICE 1 : *Faîtes un programme pour mettre une couleur encre. Son code sera introduit dans 32501 et le programme débutera à 32526.*

Le défilement des attributs

Une autre application intéressante au point de vue graphique est constituée par le défilement des attributs. Il est judicieux, dans ce cas, de faire appel à des instructions de type répétitif. Dans le programme ci-dessous proposé, c'est ainsi LDI qui occupe une position centrale

Les attributs sont décalés de droite à gauche, puis de gauche à droite. Pour nos besoins, entrez 149 comme nombre d'octets que comporte le programme.

```
32450 21,00,58   LD HL,22528   : fichier attributs
32453 E5         PUSH HL
32454 D1         POP DE
32455 06,18      LD B,24
32457 C5      ┌─►PUSH BC
32458 7E      │  LD A,(HL)     : pour rotation
              │                  complète
32459 23      │  INC HL
32460 01,00,20│  LD BC,8192    : B=32 et C=0
32463 ED,A0   │ ┌►LDI          : B=31 et C=255
32465 10,FC   │ └─DJNZ -4+2    : boucle 31 fois
32467 12      │  LD (DE),A
32468 13      │  INC DE
32469 C1      │  POP BC
32470 10,F1   └──DJNZ -15+2
32472 C9         RET
```

On a employé LDI pour avoir à la fois INC HL et INC DE, mais comme LDI fait aussi DEC BC, on a chargé BC de telle sorte que DEC BC influence C et DJNZ influence B.

Nous intercalons, maintenant, le programme de l'exemple 9 du chapitre II.

```
32473 21,00,19   LD HL,6400
32476 11,00,58   LD DE,22528
32479 01,00,03   LD BC,768
32482 ED,B0      LDIR
32484 C9         RET
```

Entrons, suivant le même principe, un programme pour défiler de gauche à droite :

```
32485 21,FF,5A   LD HL,23295
32488 E5         PUSH HL
32489 D1         POP DE
32490 06,18      LD B,24
32492 C5      ┌─►PUSH BC
32493 7E      │  LD A,(HL)
32494 2B      │  DEC HL
32495 01,00,20└─ LD BC,8192
```

```
32498 ED,A8    ↑┌►LDD
32500 10,FC    ||└─DJNZ -4+2
32502 12       ||  LD (DE),A
32503 1B       ||  DEC DE
32504 C1       ||  POP BC
32505 10,F1    |└──DJNZ -15+2
32507 C9       |   RET
```

Pour reprendre le chargement à 32485, faîtes LET a=32485, LET a$="" et GO TO 1050.

Pour expérimenter, faîtes le programme Basic :

```
10 RANDOMIZE USR 32473
20 FOR n=1 TO 32
30 RANDOMIZE USR 32450
40 NEXT n
50 FOR n=1 TO 64
60 RANDOMIZE USR 32485
70 NEXT n
999 STOP
```

EXERCICE 2 - *Ecrire un programme pour faire défiler les attributs de bas en haut, puis un autre pour défiler de haut en bas. Vous devez réaliser ces programmes uniquement avec des mnémoniques LD.*
De façon à faire des rotations complètes, il faut réserver une zone de 32 octets pour stocker une ligne d'attributs. Commencez donc votre programme à 32540.

LE SON

Avant de donner des exemples de production de sons à l'aide de C.M., nous examinons les appels de routines à l'intérieur d'une autre routine et les instructions d'échanges avec les périphériques. Les appels et les accès de routines et de périphériques sont des éléments très importants de la programmation en C.M. : les retour au Basic sont moins fréquents. Cet examen préalable des appels de routines MEM est nécessité par les caractéristiques particulières du son sur le ZX-Spectrum.

Les appels de routines

CD,qq,pp	CALL pq	GO SUB pq

exécute plusieurs instructions :

(SP-1) = PChaut	POKE SP-1, PChaut
(SP-2) = PCbas	POKE SP-2, PCbas

```
SP = SP-2              LET SP = SP-2
PC = pq                LET PC = pq
```

Dans un programme en C.M., un CALL pq provoque un saut à une routine dont l'adresse de départ est pq ; cette routine *se termine par un RET qui provoque un retour dans votre programme.*

C9 RET RETURN

exécute les instructions :

```
PCbas  = (SP)          LET PCbas  = PEEK SP
PChaut = (SP+1)        LET PChaut = PEEK (SP+1)
SP     = SP+2          LET SP = SP+2
```

CALL et RET n'ont pas d'influence sur les indicateurs. RET ramène à l'adresse qui suit le CALL puisque cette adresse est poussée dans la pile.

La routine, appelée par CALL, peut contenir des PUSH et des POP avant le RET. On peut altérer ainsi l'adresse de retour en faisant :

```
POP HL
INC HL
PUSH HL
```

qui augmente l'adresse de retour de 1, ou :

```
POP HL
LD HL,32000
PUSH HL
```

qui ramène à 32000.

Il est possible, également, de ne pas employer CALL, mais JP ou JR et, avant RET, de manipuler l'adresse de retour ; on peut aussi stocker des octets de données après un CALL et les exploiter par :

```
POP HL
LD A,(HL)
INC HL
PUSH HL
```

On aura la donnée dans A et le retour est actualisé. Cette technique est employée dans l'exemple de production des sons.

Entrée et sortie de périphérique

De même que les octets mémoire ont une adresse, au moyen de laquelle on manipule ces octets, de même les périphériques ont une adresse dont l'étendue possible

est de 65535. Le périphérique est également appelé un *port*. Les mnémoniques LD servent à charger et à reprendre des octets en mémoire et les ménmoniques IN et OUT agissent de même vis-à-vis des octets d'un périphérique.

```
DB,pp          IN A,(p)      LET A = PEEK p
```

exécute A = (p).

Le contenu de l'adresse du port p est chargé dans A.

```
D3,pp          OUT (p),A     POKE p,A
```

effectue (p) = A.

Le contenu de l'adresse du port p est chargé par A.

L'adresse du port est Ap où A est l'octet haut et p l'octet bas. Ces deux C.M. n'ont pas d'effet sur les indicateurs.

On réalise également des échanges avec les autres registres simples :

```
ED,40          IN B,(C)      LET B = PEEK BC
ED,41          OUT (C),B     POKE BC,B
```

Voir les autres C.M. en annexe. BC donne l'adresse du port dont le contenu entre dans un registre simple R, ou est chargé par ce registre R.

Les OUT (C),R n'ont pas d'effet sur les indicateurs, alors que les IN R,(C) agissent sur S, Z et P/V - *pas sur C_i*.

On réalise les échanges, entre périphériques et octets mémoire, avec les instructions suivantes :

```
ED,A2          INI
```

qui équivaut aux instructions :

```
               (HL) = (C)
               HL   = HL+1
               B    = B-1
ED,AA          IND
```

qui correspond à

```
               (HL) = (C)
               HL   = HL-1
               B    = B-1
```

Ces deux instructions agissent sur l'indicateur Z comme suit :

```
Z=1 si B=0 après exécution, si non Z=0
ED,B2          INIR
```

agit comme INI, mais avec répétition jusqu'à ce que B=0.

```
     ED,BA              INDR
```

a la même action IND, mais avec répétition jusqu'à ce que B=0. L'indicateur Z vient à 1 évidemment.

Ces deux instructions réalisent l'entrée de blocs d'octets et seront donc très importantes quand on aura les disquettes.

```
     ED,A3              OUTI
exécute                 (C) = (HL)
                        HL  = HL+1
                        B   = B-1
     ED,AB              OUTD
effectue                (C) = (HL)
                        HL  = HL-1
                        B   = B-1
     ED,B3              OTIR
```

opère comme OUTI, mais avec répétition jusqu'à ce que B=0.

```
     ED,BB              OTDR
```

fonctionne comme OUTD, mais avec répétition jusqu'à ce que B=0.

Ces instructions agissent de la même manière sur Z.

Les instructions d'entrée et de sortie avec répétition sont aussi puissantes que leurs homologues qui influent sur LD et CP.

Elles seront très utiles avec la venue prochaine des disquettes. La sauvegarde des programmes, le transfert de données, la gestion de fichiers, etc. se feront à grande vitesse.

La production de sons par l'ordinateur

Le port de sortie d'adresse 254d, FEh, commande le H.P. (bit 4), le micro (bit 3) et la couleur des bords de l'écran (bits 2, 1 et 0). Les trois bits hauts n'ont pas d'importance.

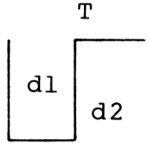


Figure 1 - une période d'un son

Pour actionner le H.P., il faut mettre le bit 4 à 0, puis à 1, pour réaliser une période T. Le nombre de périodes par seconde est la fréquence *f* et représente la hauteur du son. Pour produire un son de fréquence *f*, il faut mettre le bit 4 à 0 pendant un temps d1, puis mettre le bit 4 à 1 pendant le temps d2. Nous faisons d1 = d2 par symétrie, mais on peut essayer des durées dissymétriques qui modifient le timbre du son. Vous pouvez agir sur le bit 3 au lieu du bit 4 et envoyer ainsi des signaux par la prise micro à laquelle vous raccordez un ampli, ou à la prise micro d'un enregistreur.

Voici un petit programme qui démontre la production de sons. La 1ère ligne LD A,8 met le bit 3 à 1, ce qui met la sortie micro à 1 quand on ne l'exploite pas. Les autres bits sont à 0, donc les bords de l'écran seront noirs. Plus loin, nous employons l'instruction XOR 16d qui est un bon exemple de l'emploi de XOR. En vous souvenant des règles du chapitre I, vous constatez que, quand le bit 4 est à 1, XOR 16 l'amène à 0 et laisse les autres bits inchangés et quand, au passage suivant, le bit 4 de A est à 0, XOR 16 l'amène à 1. Deux passages forment donc une période de son.

La façon de créer des délais est examinée au paragraphe suivant.

```
32500 3E,08       LD A,8
32502 01,00,02    LD BC,512      : durée du son
32505 11,80,00 ┌─►LD DE,128      : fréquence du son
32508 67       │  LD H,A        : sauve A
32509 1B       │  DEC DE        : délai
32510 7A       │  LD A,D
32511 B3       │  OR E
32512 7C       │  LD A,H        : restaure A
32513 20,F9    └─ JR NZ, -7+2
32515 EE,10       XOR 16         : change le bit 4 de A
32517 D3,FE       OUT (FE),A     : vers le H.P.
32519 67          LD H,A         : sauve A
32520 0B          DEC BC         : délai
32521 78          LD A,B
32522 B1          OR C
32523 7C          LD A,H         : restaure A
32524 20,EB    ┌─ JR NZ, -21+2
32526 C9       └─ RET
```

et le Basic d'exploitation :

```
 10 FOR n=128 TO 4 STEP -4
 20 RANDOMIZE USR 32500
 30 POKE 32506,n
 40 NEXT n
999 STOP
```

qui change 31 fois (mettez bien TO 4 dans la ligne 10) la fréquence du son en diminuant la période T, donc en augmentant la fréquence *f*. La durée du son, déterminée par BC, ne change pas en valeur, mais bien en durée réelle. En effet, la durée est de 256 fois une période T, comme celle-ci diminue constamment, la durée également.

Si vous désirez conserver la couleur des bords que vous aviez programmée dans une déclaration Basic, il faut manipuler la variable système BORDCR en extrayant la couleur choisie et en l'introduisant dans A. Pour cela, ajoutez les C.M. suivants au début du programme (n'oubliez pas de changer la ligne Basic 30 qui charge DE).

```
3A,48,5C        LD A,(5C48)   : (BORDCR) code couleur x8
E6,38           AND 38h       : isole la couleur
OF              RRCA          : pour mettre les bits
                                5,4,3 dans 2,1,0
OF              RRCA
OF              RRCA
F6,08           OR 8          : met le bit 3 à 1
```

puis continuez par :

```
01,00,02        LD BC,512
```

etc.

Une routine plus élaborée, pour produire un son, se trouve dans la MEM à l'adresse 3B5 -949d-, exploitons-là. Cette routine nécessite que l'on charge HL et DE avant l'appel. HL donne la fréquence (=0 très aigu, =FFFF très grave). DE donne la durée, mais celle-ci dépend de la fréquence, comme dans la routine donnée ci-dessus.

Voici un exemple :

```
32500 21,90,06   LD HL,1680d  : do3
32503 11,C0,00   LD DE,192d   : durée 3/4 de sec.
32506 CD,B5,03   CALL 0385    : routine que l'on
                                appelle SON
32509 21,37,05   LD HL,1335   : mi3
32512 11,50,00   LD DE,80     : durée 1/4 de sec.
32515 CD,B5,03   CALL SON
32518 21,60,04   LD HL,1120   : sol3
32521 11,80,01   LD DE,384    : durée 1 seconde
32524 CD,B5,03   CALL SON
32527 C9         RET
```

(RANDOMIZE USR 32500)

Il faut 9 octets pour chaque son produit. Cela consomme beaucoup d'octets s'il y a beaucoup de sons. Or, on ne change chaque fois que 4 octets. L'exemple suivant, très

important car il montre la manipulation de l'adresse de retour par une sous-routine, économise 5 octets par son après la production de 4 à 5 sons. AUparavant, disons quelques mots sur la production des notes de musique.

La musique

Note	*Fréquence f*	*Période T*	*HL*	*Remarques*
do 3	256	3906	1680	La fréquence f est le nombre de périodes T par seconde.
do# 3	271	3690	1587	
ré 3	287	3484	1498	
ré# 3	304	3289	1415	La période T est exprimée en microseconde.
mi 3	322	3105	1335	
fa 3	342	2924	1258	
fa# 3	362	2762	1188	HL = T/2,325
sol	384	2604	1120	
sol #	406	2463	1059	La fréquence double à chaque octave : donc : do 4 = 2 x do 3, ré 4 = 2 x ré 3, etc.
la	430	2326	1000	
la #	456	2193	943	
si	483	2070	890	
do 4	512	1953	840	

Tableau 1 - les 12 notes de la gamme

Nous avons pris les fréquences utilisées par les physiciens qui donnent, à *do*, les puissances successives de 2 (informatique oblige).

Pour trouver les 12 Notes de la gamme, on multiplie chaque fois par la racine 12ème de 2. Ceci nous amène à une fréquence de 430 pour le *la* 3 (milieu du piano). Le *la* du diapason a une fréquence de 440 et est fixée à cette valeur depuis plusieurs décennies : nous n'en sommes pas loin ; si vous vous faîtes accompagner par un violon, celui-ci n'aura qu'à s'accorder à votre *la* (pour la petite histoire, la fréquence officielle du *la* a varié de plus de 100 hertz au cours des siècles).

Puisque *do* 3 a une fréquence de 256 cycles par seconde, il faut faire DE=256 pour une durée d'une seconde. Pour la même durée sur le *do* 4, il faut DE=512.

Exemple : "Production" musicale : extrait de Don Juan de Mozart. Nous employons, pour la première fois, la colonne étiquette dans de programme.

```
32500 C1    MUSI  POP BC
32501 0A          LD A,(BC)     : retire un octet de DEF
32502 67          LD H,A        : un octet dans H
32503 03          INC BC
32504 FE,FF       CP 255        : A-255 (octet de FIN)
32506 20,02       JR,NZ, +2+2
32508 C5          PUSH BC       : actualise la pile
32509 C9          RET           : si Z=1, retour
32510 0A          LD A,(BC)
32511 6F          LD L,A        : 1 octet dans L
32512 03          INC BC
32513 0A          LD A,(BC)
32514 57          LD D,A        : 1 octet dans D
32515 03          INC BC
32516 0A          LD A,(BC)
32517 5F          LD E,A        : 1 octet dans E
32518 03          INC BC
32519 C5          PUSH BC       : actualise la pile
32520 CD,B5,03    CALL SON
32523 18,E7       JR -25+2
          ENTREE
32525 CD,F4,7E    CALL MUSI
32528 06,90,00,C0,05,37,00,50,04,60,01,80
                  DEF           : 3 notes et 3 durées
32540 05,37,00,F0,04,60,00,60,03,48,02,00
                  DEF           : 3 notes et 3 durées
32552 04,60,01,20,03,48,00,80,02,9B,02,84
                  DEF           : 3 notes et 3 durées
32564 03,48,
      02,00       DEF           : 1 note  et 1 durée
32568 FF          DEF FIN
32569 C9          RET
              (RANDOMISE USR 32525)
```

L'entrée, située à 32525, appelle directement la sous-routine MUSI à 32500. A ce moment, l'adresse de retour, 32528 est placée dans la pile.

La sous-routine MUSI extrait cette adresse dans BC, en prend son contenu, puis fait pointer BC sur l'adresse suivante, soit 32529. Après avoir extrait 4 octets et les avoir placés successivement dans H, L, D et E, on appelle la routine SON de la MEM, puis on retourne à MUSI. Tous les 4 octets, soit le 1er, le 5ème, le 9ème, etc., on teste si le contenu est 255 qui est l'indicateur choisi pour la fin du message musical. Si A=255, on

pousse dans la pile l'adresse de retour 32569, puis RET retourne à la routine ENTREE. On a placé également RET à 32569, mais le programme peut se poursuivre avec les instructions suivantes. Il y a très peu de chance, aussi évitez-le, d'employer 255 comme octet qui charge H, car cela produit un son très grave qui se résume à des clics. C'est pour cette raison qu'on l'a choisi comme Fin du morceau musical. Vous pouvez, bien sûr, employer 255 comme 2ème, 3ème ou 4ème octet.

CORRIGÉS DES EXERCICES

EXERCICE 4-1

```
32526 21,00,58   LD HL,22528
32529 3A,F5,7E   LD A,(32501) : stocke l'encre
32532 E6,07      AND 7
32534 47         LD B,A
32535 7E      ┌─►LD A,(HL)
32536 E6,F8   │  AND F8        : masque l'encre
32538 B0      │  OR B          : met l'encre choisie
32539 77      │  LD (HL),A
32540 23      │  INC HL
32541 3E,5B   │  LD A,91d
32543 94      │  SUB A,H
32544 20,F5   └──JR NZ, -11+2
32546 C9         RET
```

Pour essayer, changez les lignes Basic :

```
50 POKE 32501,n
60 RANDOMIZE USR 32526
```

Imaginez un programme Basic pour le mélange des deux.

EXERCICE 4-2 - de bas en haut

```
32508 à 32539    DEF           : réserve de 32 octets
                                 pour stockage provi-
32540 21,00,58   LD HL,22528     soire
32543 11,FC,7E   LD DE,32508   : réserve
32546 01,20,00   LD BC,32
32549 ED,B0      LDIR          : stocke la 1ère ligne
                                 d'attributs
```

```
32551 11,00,58   LD DE,22528
32554 01,E0,02   LD BC,736
32557 ED,B0      LDIR          : transfère les lignes
32559 21,FC,7E   LD HL,32508     2-24 dans 1-23
32562 01,20,00   LD BC,32
32565 ED,B0      LDIR          : met la lère ligne
32567 C9         RET             dans la 24ème
```

et de haut en bas :

```
32568 21,FF,5A   LD HL,23295
32571 11,1B,7F   LD DE,32539
21574 01,20,00   LD BC,32
32577 ED,B8      LDDR          : stocke la 24ème ligne
32579 11,FF,5A   LD DE,23295
32582 01,E0,02   LD BC,736
32585 ED,B8      LDDR          : transfère les lignes
32587 21,1B,7F   LD HL,32539     23-1 dans 24-2
32590 01,20,00   LD BC,32
32593 ED,B8      LDDR          : met la ligne 24
32595 C9         RET             dans la lère
```

Ajoutez les lignes Basic :

```
 80 FOR n-1 TO 64
 90 RANDOMIZE USR 32540
100 NEXT n
110 FOR n=1 TO 50
120 RANDOMIZE USR 32568
130 NEXT n
140 GO TO 20
```

et voyez.

Pour arrêter, faîtes BREAK, évidemment.

CHAPITRE V

TECHNIQUES AVANCÉES APPLIQUANT LE CLAVIER ET L'AFFICHAGE

Dans la plupart des programmations interactives, logiciels de jeu notamment, pour lesquelles le ZX-Spectrum est particulièrement adapté, est mise en oeuvre une correspondance étroite entre le clavier et l'affichage. L'étude solidaire de la structure du clavier et de celle de l'écran conduit ainsi directement à des gains de vitesse considérables, en même temps qu'elle constitue une bonne introduction aux techniques de programmation avancée en langage-machine. Le présent chapitre se propose donc d'aborder successivement, en trois points : le clavier, l'affichage et la synthèse de ces deux éléments, dans un programme illustratif développé.

LE CLAVIER

L'analyse du clavier par le système

Les touches du clavier sont raccordées horizontalement, 5 par 5 et, verticalement, 8 par 8, de la façon suivante :

a - horizontalement

CAPS SHIFT, Z, X, C, V	Liaison à A8 du bus d'adresses
A, S, D, F, G	Liaison à A9 du bus d'adresses
Q, W, E, R, T	Liaison à A10 du bus d'adresses
1, 2, 3, 4, 5	Liaison à A11 du bus d'adresses
Ø, 9, 8, 7, 6	Liaison à A12 du bus d'adresses
P, O, I, U, Y	Liaison à A13 du bus d'adresses
ENTER, L, K, J, H	Liaison à A14 du bus d'adresses
SPACE, SYMBOL SHIFT, M, N, B	Liaison à A15 du bus d'adresses

b - verticalement

CAPS SHIFT, A, Q, 1, Ø, P, ENTER, SPACE	Liaison à B0
Z, S, W, 2, 9, 0, L, SYMBOL SHIFT	Liaison à B1
X, D, E, 3, 8, I, K, M	Liaison à B2
C, F, R, 4, 7, U, J, N	Liaison à B3
V, G, T, 5, 6, Y, H, B	Liaison à B4

B0 à B4 sont raccordés au C.I. 1 qui contient la logique Sinclair (figure 1, chapitre I) et, de là, dans un registre du microprocesseur, selon l'instruction IN employée.

Utilisation directe de la routine MEM du clavier

Nous trouvons, dans la MEM, une routine de lecture du clavier à 654d -28Eh. Cette routine initialise BC avec FEFE, puis appelle l'instruction IN A,(C), point de départ d'une boucle qui contient une série d'instructions dont la dernière est RLC B. Cette instruction de rotation à gauche amène successivement chaque bit de B à 0 (au départ, B = FE) ; donc B aura, pour valeurs successives, FE, FD, FB, F7, EF, DF, BF et 7F. L'instruction IN A,(C) place BC sur le bus d'adresses, puis transfère, dans A, le contenu des 8 ports successifs. Comme B est l'octet haut, on aura successivement les lignes d'adresses A8, A9, etc... à 0.

Normalement, les lignes B0 à B4 sont à 1, sauf quand on pousse sur une touche de la ligne horizontale "scrutée" à ce moment.

Nous avons donc un moyen de tester si une touche est pressée en sélectionnant l'un des 8 ports d'entrée, puis en faisant de 1 à 5 rotations pour amener le bit concerné dans C_i et, ensuite, faire un test sur C_i.

Exemple 1 - Test d'enfoncement d'une touche

a - Nous voulons savoir si la touche SPACE est pressée et, si oui, commander l'exécution d'un programme musical PROMU.

```
01,FE,7F        LD BC,7FFE      : 8ème port
                                  d'entrée clavier
ED,78           IN A,(C)        : saisie
1F              RRA             : met A0 dans Ci
D2,qq,pp        JP NC,PROMU     : saut PROMU si
                suite             SPACE est pressée
```

b - Si la touche 8 est pressée, nous voulons exécuter un programme qui fait défiler l'écran à droite, programme appelé DEFDRO.

```
01,FE,EF          LD BC,EFFE      : 5ème port d'entrée
ED,78             IN A,(C)        : saisie
1F                RRA             : met A2 dans Ci
1F                RRA
1F                RRA
D2,qq,pp          JP NC,DEFDRO    : saut à DEFDRO si
                  suite             8 est pressée
```

c - Si BREAK est pressée, le déroulement du programme s'arrête.

```
01,FE,7F          LD BC,7FFE
ED,78             IN A,(C)
1F                RRA
38,09             JR C, +9+2      : saut si SPACE
                                    n'est pas pressée
01,FE,FE          LD BC,FEFE      : si SPACE est
ED,78             IN A,(C)          pressée, on
1F                RRA               continue
D2,qq,pp          JP NC,ARRET     : saut si CAPS
                                    SHIFT est pressée
                  suite           : suite si, ni
                                    SPACE, ni CAPS
                                    SHIFT ne sont
                                    pressées
```

Vous trouverez une routine similaire de 12 octets dans la MEM à 8020d.

Passage par l'intermédiaire de la variable système LAST-K

Si la recherche de la vitesse n'est pas primordiale, on utilise alors les variables système. La variable système LAST-K contient le code de la dernière touche pressée, jusqu'à ce qu'une nouvelle touche soit pressée. Cette particularité vous permet des répétitions sans devoir laisser votre doigt sur la touche.

Le bit 5 de la variable système FLAGS se met à 1 si une nouvelle touche est pressée.

Le microprocesseur exécute 50 fois par seconde une routine qui actualise le compteur d'images, variable système FRAMES, et qui scrute le clavier en actualisant les variables système FLAGS et LAST-K s'il y a lieu. (Cette routine ne s'exécute pas quand on produit des sons.) Puisque l'ordinateur scrute le clavier toutes les 20 millisecondes, il nous suffit de tester les variables

système pour connaître la touche pressée, mais, ici, il s'agit de son code.

Reprenons l'exemple 2b ci-dessous :

```
97              SUB A,A          : met A à 0
21,3B,5C        LD HL,23611      : FLAGS
CB,6E           BIT 5,(HL)
28,0A           JR Z, +10+2      : saut si aucune
                                   touche n'est
                                   pressée
3A,08,5C        LD A,(23560)     : (LAST-K)
CB,AE           RES 5,(HL)       : remet à 0 le bit
                                   5 de FLAGS
FE,38           CP 56d           : code de 8
CA,qq,pp        JP Z,DEFDRO      : saut si 8 est
                                   pressée
                suite
```

C'est un peu plus long que l'exemple 2b mais, ici, A contient 0 si aucune touche n'est pressée ou le code de la touche pressée. Ceci permet de faire plusieurs comparaisons et de prendre les décisions adéquates. Par exemple :

```
FE,38           CP 56d           : code de 8
CA,qq,pp        JP Z,DEFDRO      : défiler à droite
FE,35           CP 53d           : code de 5
CA,yy,xx        JP Z,DEFGAU      : défiler à gauche
                etc.
```

Les pauses

Les pauses illimitées, qui attendent l'enfoncement d'une touche pour prendre une décision, ou les pauses à durée limitée pour ralentir ou faire varier le déroulement d'un programme en C.M., sont essentielles dans les jeux interactifs et dans un grand nombre d'autres applications.

Equivalent du Basic INPUT

Réalisons une pause prolongée jusqu'à enfoncement d'une touche :

```
21,3B,5C        LD HL,23611
CB,6E           BIT 5,(HL)       : Z = (HL)5 barre
28,FC           JR Z, -4+2
```

La boucle s'exécute jusqu'à ce qu'une touche soit pressée.

Si on poursuit par :

```
3A,08,5C        LD A,(23560)
CB,AE           RES 5,(HL)
```

on réalise ainsi la déclaration Basic INPUT qui attend une entrée avant de poursuivre.

Equivalent du Basic INKEY$

En revanche, si on veut réaliser la fonction Basic INKEY$ qui, elle, n'interrompt pas le déroulement d'un programme, mais saisit une touche à un certain moment, il suffit de reprendre l'avant-dernier exemple tel quel (SUB A,A, etc...).

On peut également utiliser la routine qui se trouve à 654d -28Eh-. Cette routine retourne une valeur dans DE (voir chapitre 7). E = 255 si aucune touche n'est pressée.

Les pauses illimitées

Pour réaliser une pause tant qu'une touche n'est pas pressée :

```
CD,8E,02        ┌► CALL 028E      : scrute
1C              │  INC E
28,FA           └─ JR Z, -6+2
```

donc, bouclage jusqu'à enfoncement d'une touche, et :

```
CD,8E,02        ┌► CALL 028E      : scrute
1C              │  INC E
20,FA           └─ JR NZ, -6+2
```

bouclage jusqu'à ce qu'on ne pousse plus sur une touche.

Les pauses limitées

Pour faire des pauses à temps limité, nous employons le temps nécessaire que met l'ordinateur pour exécuter une ou plusieurs instructions. Chaque instruction dure un certain nombre de cycles, ou périodes, d'horloge. Sachant qu'une période d'horloge dure 0,2857 microsec. (voir chapitre I), le délai se calcule avec précision. Nous donnons, ci-après, le nombre de cycles d'horloge pour les instructions les plus couramment employées pour élaborer un délai.

```
06,20           LD B,32d        : 7T
10,FE           ⊆DJNZ -2+2      : 13T quand B ≠ 0
                                  8T quand B = 0
```

Boucle 32 fois sur lui-même.

Cette pause dure 0,2857 x (7+31x13+1x8) = 119,4 µ sec.

On peut intercaler une instruction dans la boucle pour augmenter le temps.

```
06,20              LD B,32d        :  7T
FD,21,3A,5C      ┌→LD IY,23610     : 14T
10,FA            └─DJNZ -6+2       : 13T pour B ≠ 0
                                   :  8T pour B = 0
```

ce qui ajoute 32 fois 14T, soit 448 T ou 128 microsec.

Deux boucles imbriquées sont possibles également :

```
06,30              LD B,48d        :  7T
0E,FF            ┌→LD C,255d       :  7T
0D               │┌→DEC C          :  4T
20,FD            └─JR NZ, -3+2     : 12T si NZ est vrai
                                      7T si NZ est faux
10,F9            └─DJNZ -7+2       : 13T si B ≠ 0
                                      8T si B = 0
```

La boucle C dure 254 x 16 + 11 = 4075T.

La boucle totale dure 7 + (7 + 4075 + 13) x 47 + (7 + 4075 + 8) = 196562T ou 56158 microsec. ou 56 msec.

L'emploi d'un registre double amène :

```
21,00,08           LD HL,2048d     : 10T
2B               ┌→DEC HL          :  6T
7C               │ LD A,H          :  4T
B5               │ OR L            :  4T
20,FB            └─JR NZ, -5+2     : 12T si Z = 0
                                      7T si Z = 1
```

qui dure 10 + 26 x 2047 + 21 = 53253T ou 15214 microsec.

Comme DEC HL n'influence pas les indicateurs, on a fait LD A,H, puis OR L qui met Z à 1 quand HL = 0. Nous avons, d'ailleurs, employé cette méthode pour fixer la fréquence des sons dans l'exemple du développement précédent consacré au son.

Exemple 2 - Voici un programme qui met plusieurs couleurs à la fois sur les bords de l'écran. Ceci pour montrer la synchronisation de l'image avec l'instruction HALT, ce qui permet de remplir toujours à partir du haut de l'écran. On emploie les pauses programmées variables, l'appel de sous-routine, les entrées/sorties et l'attente d'une touche. La durée du programme doit être plus petite que 20 msec., sinon la synchro-trame, qui apparaît 50 fois par seconde, se ferait dans le programme.

```
          BORD
32500 21,D4,00     LD HL,212
32503 D3,FE        OUT (FE),A
32505 2B         ┌►DEC HL
32506 7C         │ LD A,H
32507 B5         │ OR L
32508 20,FB      └─JR NZ, -5+2
32510 C9           RET
32511 76   ENT  ┌►HALT
32512 3E,00     │ LD A,0
32514 21,CE,01  │ LD HL,462
32517 CD,F7,7E  │ CALL 32503
32520 06,06     │ LD B,6
32522 0E,00     │ LD C,0
32524 0C        ├►INC C
32525 79        │ LD A,C
32526 CD,F4,7E  │ CALL BORD
32529 10,F9     └─DJNZ -7+2
32531 0C          INC C
32532 79          LD A,C
32533 D3,FE       OUT (FE),A
32535 01,FE,7F    LD BC,7FFE
32538 ED,78       IN A,(C)
32540 1F          RRA              : teste SPACE
32541 38,E0     └─JR C, -32+2
32543 C9          RET
```

```
RANDOMIZE USR 32511
```

L'AFFICHAGE

Structure des fichiers d'affichage et d'attributs

En préalable aux explications qui suivent, entrer le Basic suivant :

```
 10 FOR n=16384 TO 22527
 20 POKE n;255
 30 NEXT n
999 STOP
```

Ligne		*Début de ligne*		*Fin de ligne*	
		Affichage	*Attribut*	*Affichage*	*Attribut*
ligne 0	1	16384		16415	
	2	16640		16671	
	3	16896		16927	
	4	17152		17183	
	5	17408	22528	17439	22559
	6	17664		17695	
	7	17920		17951	
	8	18176		18207	
ligne 1	1	16416		16447	
	...		22560	...	22591
	8	18208		18239	
ligne 2	1	16448		16479	
	...		22592	...	22623
	8	18240		18271	
ligne 3	1	16480		16511	
	...		22624	...	22655
	8	18272		18303	
ligne 4	1	16512		16543	
	...		22656	...	22687
	8	18304		18335	
ligne 5	1	16544		16575	
	...		22688	...	22719
	8	18336		18367	
ligne 6	1	16576		16607	
	...		22720	...	22751
	8	18368		18399	
ligne 7	1	16608		16639	
	...		22752	...	22783
	8	18400		18431	
ligne 8	1	18432	22784	18463	22815
ligne 9	1	18464	22816	18495	22847
ligne 10	1	18496	22848	18527	22879
ligne 11	1	18528	22880	18559	22911
ligne 12	1	18560	22912	18591	22943
ligne 13	1	18592	22944	18623	22975
ligne 14	1	18624	22976	18655	23007
ligne 15	1	18656	23008	18687	23039

→

Ligne		*Début de ligne*		*Fin de ligne*	
		Affichage	*Attribut*	*Affichage*	*Attribut*
ligne 16	1	20480	23040	20511	23071
ligne 17	1	20512	23072	20543	23103
ligne 18	1	20544	23104	20575	23135
ligne 19	1	20576	23136	20607	23167
ligne 20	1	20608	23168	20639	23199
ligne 21	1	20640	23200	20671	23231
ligne 22	1	20672	23232	20703	23263
ligne 23	1	20704		20735	
		...	23264	...	23295
	8	22496		22527	

Tableau 2 - Fichiers d'affichage et d'attributs

L'affichage de l'écran est divisé en trois zones de 8 lignes. En observant la succession numérique des octets, on constate que les trois zones se remplissent successivement et identiquement. Examinons le remplissage de la première zone. Dans l'ordre successif des octets, on remplit la ligne des premiers octets de la première ligne, puis la ligne des premiers octets de la deuxième ligne, puis la ligne des premiers octets de la troisième ligne, et ainsi de suite jusqu'à 16639. Ensuite, les lignes des deuxièmes octets des lignes 1 à 8, puis les lignes des troisièmes octets des lignes de 1 à 8, et ainsi de suite jusqu'à 18431. Ensuite, on emploie le même procédé pour la zone 2, de 18432 à 20479, puis pour la zone 3, de 20480 à 22527.

Nous avons employé le mot ligne dans deux sens différents : la ligne des premiers octets - concerne la ligne des pixels - de la première ligne - concerne la ligne des caractères -. La confusion est possible ; pour l'éviter, nous dirons la position OCTET pour désigner l'octet dans une ligne de pixels ; la position PRINT pour désigner la position d'un caractère qui comporte 8 positions OCTET ; la position PLOT pour désigner un pixel d'une position OCTET.

Examinez attentivement le tableau 2 pour saisir cette succession d'octets et faîtes le petit programme Basic précédent pour le visualiser. Nous avons représenté les 8 lignes OCTET pour la première ligne PRINT, ensuite, les premières et huitièmes lignes OCTET pour les deuxième à huitième ligne PRINT, ce qui termine la première zone. Pour les lignes PRINT 9 à 24, nous avons représenté une ligne OCTET, sauf pour la 24ème où il y en a deux.

Comme vous avez pu le constater à la lecture des exemples des chapitres précédents, le fichier d'attributs est constitué de 768 octets qui se succèdent numériquement. Chaque octet d'attributs affecte 64 positions PLOT ou 8 positions OCTET ou 1 position PRINT.

Posons x = 16384 début 1ère zone
y = 18432 début 2ème zone
z = 20480 début 3ème zone

et remarquons que y = x + 2048 et que z = x + 2*2048.

D'autre part, dans une ligne PRINT, par exemple la première :
- la deuxième ligne OCTET commence à x + 256 ;
- la troisième ligne OCTET commence à x + 2*256 ;
et ainsi de suite jusqu'à la 8ème ligne OCTET.

Pour passer à la 9ème ligne OCTET (début de la 2ème ligne PRINT), on a x + 32 ; la 10ème ligne OCTET commence à x + 256 + 32, et ainsi de suite.

Réalisation de différents défilement d'écran

Réalisons le défilement de bas en haut, par ligne de pixels. Au vu du fichier d'affichage, pour employer LDIR afin de transférer une ligne de pixels vers la précédente, il faut faire DE = 16384 et HL = 16640, puis LDIR avec BC = 32, ensuite DE = 16640 et HL = 16896, puis LDIR, etc... Les chargements de DE et HL changent à chaque ligne de pixels. Faisons le schéma de ces chargements en employant les paramètres a, b, c, qui sont les coefficients des nombres 2048, 256 et 32.

a	b	c		
x + 0.2048 + 0.256 + 0.32 = DE = 16384			}	ligne 2 vers 1
x + 0.2048 + 1.256 + 0.32 = HL = 16640			}	
"				
"				
x + 0.2048 + 6.256 + 0.32 = DE = 17920			}	ligne 8 vers 7
x + 0.2048 + 7.256 + 0.32 = HL = 18176			}	
x + 0.2048 + 7.256 + 0.32 = DE = 18176			}	ligne 9 vers 8
x + 0.2048 + 0.256 + 1.32 = HL = 16416			}	
quand b = 0, incrémenter c				
x + 0.2048 + 0.256 + 1.32 = DE = 16416			}	ligne 10 vers 9
x + 0.2040 + 1.256 + 1.32 = HL = 16672			}	
"				
"				

```
   a       b       c
x + 0.2048 + 6.256 + 7.32 = DE = 18144 }
x + 0.2048 + 7.256 + 7.32 = HL = 18400 } ligne 64 vers 63

x + 0.2048 + 7.256 + 7.32 = DE = 18400 }
x + 1.2048 + 0.256 + 0.32 = HL = 18432 } ligne 65 vers 64

                quand c = 0, incrémenter a
```

Remarquons également que la valeur de DE est égale à la valeur précédente de HL.

Dans le programme qui suit, A = x + a.2048, B = b.256 et C = c.32, D = A + B et E = C, puis après manipulation de B et, éventuellement de C, H = A + B et L = C

Exemple 3 - Effectuer le défilement de l'écran de bas en haut d'une ligne de pixels.

```
32500 3E,40  D.HAUT   LD A,64       : octet haut=16384
32502 01,00,00        LD BC,0
32505 5F              LD E,A        :┐ sauve A
32506 80              ADD A,B       :│
32507 57              LD D,A        :│ D = A + B
32508 7B              LD A,E        :│ restaure A
32509 59              LD E,C        :┘ E=C  DE pointe
32510 04              INC B         : correspond à +256
32511 CB,58           BIT 3,B       : Z = B3
32513 28,0A           JR Z, +10+2   : saut si B plus
                                      petit que 8
32515 6F              LD L,A        :┐ sauve A
32516 79              LD A,C        :│
32517 C620            ADD A,32      :│ influence Z
32519 4F              LD C,A        :│ C = C + 32
32520 7D              LD A,L        :┘ restaure A
32521 20,02           JR NZ, +2+2   : saut si C est
                                      différent de 0
32523 C6,08           ADD A,8       : correspond à
                                      + 2048
32525 CB,98           RES 3,B       : B3 = 0
32527 CF              LD L,A        :┐ sauve A
32528 80              ADD A,B       :│
32529 67              LD H,A        :│ H = A + B
32530 7D              LD A,L        :│ restaure A
32531 69              LD L,C        :┘ L = C HL pointe
32532 C5              PUSH BC       :┐ sauve BC
32533 01,20,00        LD BC,32      :│
32536 ED,B0           LDIR          :│ transfère une
                                      ligne de pixels
32538 C1              POP BC        :┘ restaure BC
```

```
32539 FE,58          CP 88          : correspond à
                                      22528
32541 20,DA          JR NZ, -38+2   : saut si A plus
                                      petit que 88
32543 C9             RET
```

Ceci transfère tout l'écran d'une ligne de pixels vers le haut. Si vous avez le texte de chargement sur l'écran, faites :

RANDOMIZE USR 32500 plusieurs fois.

Ajoutons une routine pour déterminer le nombre de lignes de pixels que l'on veut voir défiler :

```
32554 06,80          LD B,128       : doit être plus
                                      petit que 193
32556 C5             PUSH BC
32557 CD,F4,7E       CALL D.HAUT
32560 C1             POP BC
32561 10,F9          DJNZ -7+2
32563 C9             RET
```

Ce programme fait défiler 128 lignes de pixels ; pour défiler un autre nombre de lignes, changer 32555.

Essayer :

```
10 PLOT 57,27 : DRAW 120,120,PI*209
20 RANDOMIZE USR 32554
999 STOP
```

Puisqu'on transfère la 193ème ligne, les 32 premiers octets du fichier d'attributs apparaissent à la 192ème ligne de pixels. Ces octets sont remplis de 56d -38h- (l'encre est noire et le papier blanc). Pour éviter cela, il faut blanchir la 192ème ligne de pixels en utilisant l'exemple 1 du chapitre III. Ajoutons les C.M. suivants :

```
32543 21,E0,57       LD HL,22496
32546 06,20          LD B,32
32548 36,00          LD (HL),0
32550 23             INC HL
32551 10,FB          DJNZ -5+2
32553 C9             RET
```

et relancer le programme Basic ci-dessus.

EXERCICE I : *En s'inspirant de l'exemple ci-avant, réaliser une programmation qui effectue le défilement de l'écran de haut en bas, de la valeur d'une ligne de pixels.*

Calcul des différentes positions d'affichage

Le **PRINT AT** x,y du Basic commence la numérotation des lignes par le haut de 0 à 23 (mais n'accepte que de 0 à 21), et la numérotation des colonnes de 0 à 31. A chaque affichage d'un caractère, les variables système, concernées par la position PRINT, sont mises à jour. En C.M., on ne met pas ces variables système à jour, mais on calcule, chaque fois, directement, la position PRINT, en vue de l'affichage d'un caractère ou d'un texte. Les routines, qui effectuent les PRINT AT et PRINT TAB, feront appel à cette routine de calcul de la position PRINT. Il y a autant de positions ATTR que de positions PRINT ; elles sont calculées simultanément.

Extrayons, des fichiers du tableau 2, les premières positions de chaque ligne de caractère.

	pos.PRINT	x=16384	pos.ATTR	y=22528
ligne 0	16384	x+0.32	22528	y+0.32
ligne 1	16416	x+1.32	22560	y+1.32
"				
ligne 7	16608	x+7.32	22752	y+7.32
ligne 8	18432	x+8.32+1792	22784	y+8.32
"				
ligne 15	18656	x+15.32+1792	23008	y+15.32
ligne 16	20480	x+16.32+3584	23040	y+16.32
"				
ligne 23	20704	x+23.32+3584	23264	y+23.32

Au vu de toutes ces positions, le programme suivant se comprend aisément.

Equivalent du ***PRINT*** *du Basic*

Exemple 4 - Calcul des positions PRINT et ATTR.

Avant l'appel de cette routine, il faut charger H avec le numéro de ligne (de 0 à 23) et L avec le numéro de colonne (de 0 à 31). Mettez 199 comme nombre d'octets de chargement.

```
32400 16,00  POS.PR   LD D,0
32402 A7              AND A          : Ci = 0
32403 7C              LD A,H
32404 06,05           LD B,5
32406 17            ┌►RLA            : multiplie par 32
32407 CB,12         │ RL D
32409 10,FB         └─DJNZ -5+2
32411 85              ADD A,L
```

```
32412 5F              LD E,A
32413 D5              PUSH DE
32414 C1              POP BC
32415 78              LD A,B
32416 C6,58           ADD A,88d      : ajoute 22528
32418 47              LD B,A         : BC pointe
32419 CB,42           BIT 0,D        : Z = D0
32421 28,04           JR Z, +4+2     : saut si Z = 1
32423 7A              LD A,D         : si D0 = 1, Z = 0
32424 C6,07           ADD A,7        : ajoute 1792
32426 57              LD D,A
32427 CB,4A           BIT 1,D        : Z = D1
32429 28,04           JR Z, +4+2     : saut si Z = 1
32431 7A              LD A,D         : si D1 = 1, Z = 0
32432 C6,0E           ADD A,14d      : ajoute 3584
32434 57              LD D,A
32435 7A              LD A,D
32436 C6,40           ADD A,64d      : ajoute 16384
32438 57              LD D,A         : DE pointe
32439 C9              RET
```

Au retour de cette routine, BC pointe sur la position ATTR et DE sur la position PRINT.

Nous n'avons fait aucun test, au contraire du Basic. On aurait pu tester si H est compris entre 0 et 23 et L entre 0 et 31 et, dans les cas contraires, provoquer un message d'erreur avec un mnémonique RST : nous verrons les RST dans le chapitre 7 . Vous devez charger HL avec une valeur convenable. Si vous prévoyez que ce chargement pourrait être erroné, ajoutez des tests.

Exemple 5 - Calcul de la position OCTET.

D'après le tableau 2, le premier octet d'une ligne est la position PRINT.

Avant l'appel de cette routine, il faut charger H avec le numéro de ligne d'octets (de 0 à 191) et L avec le numéro de colonne (de 0 à 31).

```
32440 7C  POS.OC      LD A,H
32441 E6,F8           AND F8h        : H modulo 8
                                       et Ci = 0
32443 1F              RRA
32444 1F              RRA
32445 1F              RRA            : A pointe sur le
                                       numéro de ligne
                                       pos. PRINT
32446 E5              PUSH HL        : sauve H
32447 67              LD H,A
32448 CD,90,7E        CALL POS.PR    : 32400
```

```
32451 E1              POP HL        : restaure H
32452 7C              LD A,H
32453 E6,07           AND 7         : isole numéro de
                                      ligne (0 à 7) à
                                      partir de la
                                      position octet
32455 82              ADD A,D       : ajoute de 0 à
                                      7 fois 256
32456 57              LD D,A        : DE pointe
32457 C9              RET
```

Au retour de cette routine, DE pointe sur la position OCTET et, en cas de besoin, BC pointe sur la position ATTR.

On prend d'abord les 5 bits hauts de H qu'on fait tourner vers la droite 3 fois avant d'appeler POS.PR, ensuite, en prenant les 3 bits bas de H, on pointe sur le nombre de lignes d'octets au delà de la position PRINT.

Ici aussi, on n'a fait aucun test de vérité sur HL.

*Equivalent du **PLOT** du Basic*

Exemple 6 - Calcul de la position PLOT.

Avant l'appel, on charge H avec le numéro de ligne d'octets (de 0 à 191) et L avec le numéro du pixel dans la ligne (de 0 à 255).

```
32458 7D      POS.PL  LD A,L
32459 E6,F8           AND F8h       : L modulo 8
                                      et Ci = 0
32461 1F              RRA
32462 1F              RRA
32463 1F              RRA
32464 E5              PUSH HL
32465 6F              LD L,A
32466 CD,B8,7E        CALL POS.OC
32469 E1              POP HL
32470 7D              LD A,L
32471 E6,07           AND 7
32473 C9              RET
```

Au retour, DE pointe sur la position OCTET et A sur le pixel dans cette position. Pour rappel, BC pointe sur la position ATTR. Cette façon de procéder part du coin supérieur gauche de coordonnées (0,0), selon la *figure 2*.

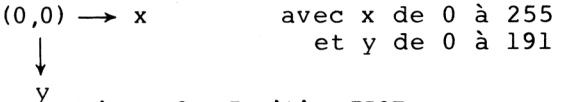


Figure 2 - Position PLOT

Si vous voulez les coordonnées cartésiennes telles qu'on les emploie avec le PLOT du Basic, il faut partir du point (0,0)en dessous à gauche. Mais, il y a un problème : vous n'avez pas la correspondance avec le PLOT du Basic, car les C.M. remplissent tout l'écran. Le départ (0,0) du Basic correspondrait à notre départ (0,16). Le Basic limite l'ordonnée à 175 qui est le dessus de l'écran.

Si vous préférez, malgré tout, les coordonnées cartésiennes, il suffira, au début du programme précédent, de faire le complément à 191 de la variable contenue dans H :

```
LD A,H
CPL
RES 6,A
LD H,A
```

RES 6,A pour retrancher 64d, sinon vous auriez le complément à 255.

Exemple 7 - Calcul de l'adresse de la matrice de caractères.

Pour les besoins de l'affichage, il est nécessaire de pointer sur le premier octet de la matrice du caractère à afficher.

Avant l'appel de la routine, A doit être chargé avec le code du caractère à afficher.

```
32474 CB,7F  MATCAR  BIT 7,A       : Z = A7
32476 28,09          JR Z, +9+2    : saut si A<128
32478 06,00          LD B,0        : si A>127
32480 4F             LD C,A
32481 21,DB,07       LD HL,2011d
32484 09             ADD HL,BC     : 2011 + code UDG
32485 18,03          JR +3+2
32487 26,00          LD H,0
32489 6F             LD L,A
32490 29             ADD HL,HL     : multiplie par 8
32491 29             ADD HL,HL
32492 29             ADD HL,HL
32493 7C             LD A,H
32494 C6,3C          ADD A,60d     : ajoute 15360
32496 67             LD H,A
32497 C9             RET
```

Au retour, HL pointe sur le premier octet de la matrice de caractères dont le code était dans A. On n'a fait aucun test sur la validité du code. Cependant, pour pouvoir afficher des caractères UDG - graphique défini par l'utilisateur - la routine commence par un test.

Si A est plus petit que 128, on met A dans HL, on multiplie par 8 et on ajoute 15360. SI A est plus grand que 127, supposons-le égal à 144 qui est le code du premier graphique ; on met 144 dans BC, on ajoute 2011, on multiplie par 8 et on ajoute 15360. Nous aurons 2155 x 8 + 15360 = 32600 qui est l'adresse du premier caractère UDG pour un ZX-Spectrum avec 16K. Si vous avez 48K, ou si vous avez changé UDG pour avoir plus de 21 caractères affichables, il faut rectifier pour finalement pointer sur UDG.

Si vous entrez un code plus petit que 32 ou de 128 à 143 ou plus grand que 164, il n'arrivera rien, sauf un affichage biscornu. Comme on emploie BC dans cette routine, si vous vous en servez dans votre programme, il faut le sauver avant l'appel puis le restaurer.

L'affichage de caractères

Nous conseillons de sauver les 4 routines précédentes à l'aide d'un SAVE "CM" CODE 32400,367.

```
POS.PR    32400 - 7E90h -
POS.OC    32440 - 7EB8h -
POS.PL    32458 - 7ECAh -
MATCAR    32474 - 7EDAh -
```

Elles ne sont pas relogeables, sauf la première et la quatrième.

A l'aide de ces routines, nous affichons n'importe quoi sur l'écran, à l'endroit souhaité.

Exemple 8 - Réalisons le Basic suivant :

```
PRINT AT x,y ; "z"
```

Avant l'appel de la routine, il faut charger x dans H, y dans L et le code z dans A.

```
32498 F5    AFF.CA   PUSH AF        : sauve A
32499 CD,90,7E       CALL POS.PR   : BC et DE pointent
32502 F1             POP AF        : restaure A
32503 C5             PUSH BC       : sauve BC
32504 CD,DA,7E       CALL MATCAR   : HL pointe
32507 06,08          LD B,8
32509 7E          ┌─►LD A,(HL)
32510 00          │  NOP
32511 00          │  NOP
32512 12          │  LD (DE),A
32513 14          │  INC D        : ajoute 256
32514 23          │  INC HL
32515 10,F8       └──DJNZ -8+2
```

```
32517 C1          POP BC        : BC pointe sur ATTR
32518 C9          RET
```

Entrez, ensuite, un petit programme d'exploitation :

```
32519 26,0C       LD H,12       : 12ème ligne
32521 2E,10       LD L,16       : 16ème colonne
32523 3E,4F       LD A,79       : code de 0
32525 CD,F2,7E    CALL AFF.CA
32528 3E,38       LD A,56       : papier blanc
32530 02          LD (BC),A
32531 C9          RET
```

(RANDOMIZE USR 32519)

Avant de poursuivre, veuillez noter que le **PRINT TAB** du Basic n'est qu'un PRINT AT x,y où on ne change que y.

Au départ de la routine, on sauve A puisque POS.PR l'utilise, puis on le restaure. De même, on sauve BC parce que MATCAR et notre routine AFF.CA l'emploie. On transfère, ensuite, les 8 octets de la matrice de caractère pointé par HL dans la position PRINT pointée par DE. Ensuite, on restaure BC pour pouvoir agir sur les attributs.

Le petit programme d'exploitation nous permet d'expérimenter en changeant les paramètres :

```
POKE 32522,17
POKE 32524,80
RANDOMIZE USR 32519
```

et vous aurez un P à côté du 0.

Vous pouvez agir sur les attributs de la position PRINT en modifiant l'octet 32529 comme suit :

- le bit 7 contrôle le clignotement ;
- le bit 6 contrôle la brillance ;
- les bits 5, 4, 3 contrôlent la couleur papier ;
- les bits 2, 1, 0 contrôlent la couleur encre.

Vous calculez la valeur de l'octet en additionnant 128 (pour clignoter) + 64 (pour briller) + 8x couleur papier + couleur encre. Vous voulez clignoter en employant du papier blanc et de l'encre noire, il vient 128 + 0 + 8 x 7 + 0 = 184

```
POKE 32529,184
```

puis RANDOMIZE USR 32519

Vous agissez donc sur l'attribut localement.

D'autre part si, dans votre Basic, vous avez déterminé les attributs de tout l'écran que vous désirez conserver, faîtes simplement POKE 32530,201 qui place un RET dans cet octet.

Il y a encore deux paramètres sur lesquels vous pouvez agir : INVERSE et OVER. Pour cette raison, il y a deux NOP dans la routine AFF.CA.

POKE 32510,47, et vous avez :

```
32509 7E          LD A,(HL)
32510 2F          CPL
32511 00          NOP
32512 12          LD (DE),A
```

qui réalise **INVERSE.**

Notez que, si vous choisissez du papier noir et de l'encre blanche dans l'attribut, vous réalisez également **INVERSE.**

Pour **OVER**, faîtes les POKE nécessaires pour avoir :

```
32509 4E          LD C,(HL)
32510 1A          LD A,(DE)
32511 A9          XOR C
32512 12          LD (DE),A
```

La programmation Basic actualise la variable système P-FLAG, adresse 23697. Les bits 0 et 1 sont à 1 pour OVER 1, et les bits 2 et 3 sont à 1 pour INVERSE 1. Ici aussi, vous pouvez employer cette variable système pour faire un choix.

Exemple 9 - Affichage d'un texte.

```
32519 C1     AFF.TE→POP BC
32520 0A             LD A,(BC)     : retire un octet
                                     de DEF
32521 67             LD H,A        : n° de ligne
32522 03             INC BC
32523 FE,FF          CP FF         : A-FF (octet FIN)
32525 20,02          JR NZ, +2+2
32527 C5             PUSH BC       : si Z = 1, actua-
                                     liser SP
32528 C9             RET
32529 0A             LD A,(BC)
32530 6F             LD L,A        : n° de colonne
32531 03             INC BC
32532 0A             LD A,(BC)     : code du caractère
32533 03             INC BC
32534 C5             PUSH BC       : actualise SP
32535 CD,F2,7E       CALL AFF.CA
```

Ici, on peut manipuler les attributs.

```
32538 18,EB              └JR -21+2
32540 CD,07,7F  ENTREE   CALL AFF.TE
32543 0C,09,4D,0C,0A,
      45,0C,0B,52        DEF      : MER
32552 0C,0C,43,0C,0D,
      49,0C,0E,20        DEF      : CI
32561 0C,0F,42,0C,10,45,
      0C,11,41,0C,12,55  DEF      : BEAU
32573 0C,13,43,0C,14,4F,
      0C,15,55,0C,16,50  DEF      : COUP
32585 FF                 DEF FIN
32586 C9                 RET
```

Faites RANDOMIZE USR 32540.

Nous avons employé la même technique pour l'exemple de production musicale du chapitre précédent.

On charge H, L et A aux valeurs convenables avant l'appel de AFF.CA. Après cette instruction, on peut agir sur les attributs car, ici, BC pointe sur la position ATTR. Dans ce cas, n'oubliez pas de changer le JR qui suit.

UN EXEMPLE SYNTHETIQUE : L'IMPRESSION DE CARACTERES AGRANDIS

Pour terminer ce chapitre, nous allons élaborer un programme pour afficher des caractères agrandis. L'agrandissement obtenu peut être deux fois, trois fois, quatre fois, etc... en hauteur, en largeur ou les deux à la fois. La distance entre deux caractères est variable également. De cette manière, en la réduisant et en choisissant d'agrandir une fois (donc sans agrandissement), nous afficherons 36 caractères sur une ligne.

L'action sur un pixel déterminé

Au préalable, comme on agit au niveau du pixel, nous avons besoin d'un programme pour afficher un pixel. D'autre part, nous supposons que vous avez sauvé les routines POS.PR, POS.OC, POS.PL et MATCAR, si non, faîtes-le avec la commande

```
SAVE "CM" CODE 32400,367
```

Ensuite, chargez le programme de chargement du chapitre I et entrez 399 comme octets demandés. Faîtes alors :

```
LOAD "CM" CODE 32200,567
```

Nous avons maintenant :

- 7DC8 de 32200 à 32239 POS.PR
- 7DF0 de 32240 à 32257 POS.OC, mais avec les changements (32249) = C8 et (32250) = 7D
- 7E02 de 32258 à 32273 POS.PL, également en changeant (32267) = FO et (32268) = 7D
- 7E12 de 32274 à 32297 MATCAR

Exemple 10 - Pour réaliser la commande PLOT.

```
32298 CD,02,7E  AFF.PL CALL POS.PL
32301 2F               CPL
32302 E6,07            AND 7
32304 17               RLA
32305 17               RLA
32306 17               RLA
32307 F6,C7            OR C7          : pour faire SET
32309 21,04,5B         LD HL,23300    : adresse qui
                                        contient un
                                        octet de matrice
32312 CB,7E            BIT 7,(HL)     : teste le bit 7
32314 20,02            JR NZ, +2+2
32316 D6,40            SUB A,40h      : pour faire RES
32318 32,43,7E         LD (32323),A
32321 1A               LD A,(DE)
32322 CB,00            SET ou RES
                       de 0 à 7,A     : 16 possibilités
32324 12               LD (DE),A
32325 C9               RET
```

Ce programme nous servira pour l'affichage de caractère agrandi. Cependant, vous pouvez l'utiliser pour réaliser la commande PLOT. Tout d'abord, un commentaire : LD (32323),A charge un octet du programme qui est le second de CB,00. Suivant le contenu de A, il modifie le mnémonique SET ou RES pour les bits 0 à 7 : un pixel s'affiche ou s'éteint. Pour bien comprendre la manipulation de A dans les 7 premières instructions, prenez une feuille et représentez les deuxièmes octets de SET 0,A à SET 7,A et de RES 0,A à RES 7,A en binaire, soit C7 à FF et 87 à BF, l'un en dessous de l'autre : vous voyez que, quand on retranche 40h des octets de SET, on retrouve les octets de RES. De plus, les bits 0 à 7, sur lesquels on agit, sont représentés dans les bits 5, 4, 3 des octets. Voilà pourquoi on effectue trois fois RLA pour les amener là et, ensuite, OR C7 pour faire SET.

Pour contrôler ce programme AFF.PL, ajoutons les C.M. suivants :

```
32326 26,00          LD H,0
32328 2E,00          LD L,0
32330 3E,00          LD A,0
32332 32,04,58       LD (23300),A
32335 CD,2A,7E       CALL AFF.PL
32338 C9             RET
```

et, pour l'exploiter, on doit charger H avec un nombre de 0 à 191 et L avec un nombre de 0 à 255 ; ensuite, on doit influencer le bit 7 de A : on charge donc 128 dans A pour afficher et 0 pour effacer.

Entrons maintenant le Basic :

```
  5 POKE 32331,128
 10 FOR n+1 TO 191
 20 POKE 32327,n: POKE 32329,n
 25 RANDOMIZE USR 32326
 30 NEXT n
 40 POKE 32331,0
 50 GO TO 10
999 STOP
```

(Presser BREAK pour sortir du programme).

Les grandes lignes du programme

Pour afficher des caractères agrandis, voici comment nous allons procéder. Posons :

- x l'abscisse (position pixel à partir de la gauche) ;
- y l'ordonnée (position pixel à partir du haut) ;
- HY la hauteur d'agrandissement ;
- LX la largeur d'agrandissement ;
- ES l'espace d'un caractère à l'autre (normalement, 8 pixels).

Supposons qu'on agrandisse deux fois en hauteur et deux fois en largeur, représentons un pixel par un petit carré.

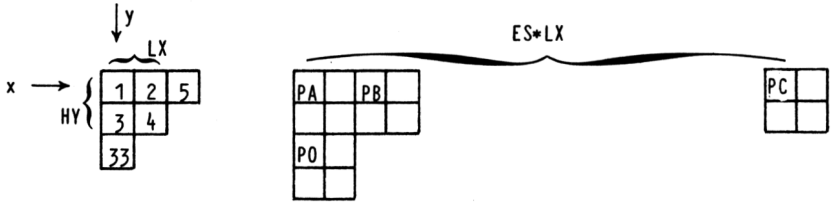


Figure 3 - Plan d'affichage des pixels

Après avoir déterminé x et y pour pointer sur 1 qui est la première position d'affichage, nous noircissons (ou blanchissons) 1 puis 2 (fonction de LX), puis 3 et 4 (fonction de HY). A ce moment, on passe à la position 5.

Nous extrayons la matrice du caractère à agrandir et posons :

- PA, position d'affichage du premier pixel ;
- PB, position du bit dans le premier octet de la matrice ;
- PO, position de l'octet : successivement les 8 octets de la matrice ;
- PC, position du caractère : autant de fois que de caractères à agrandir.

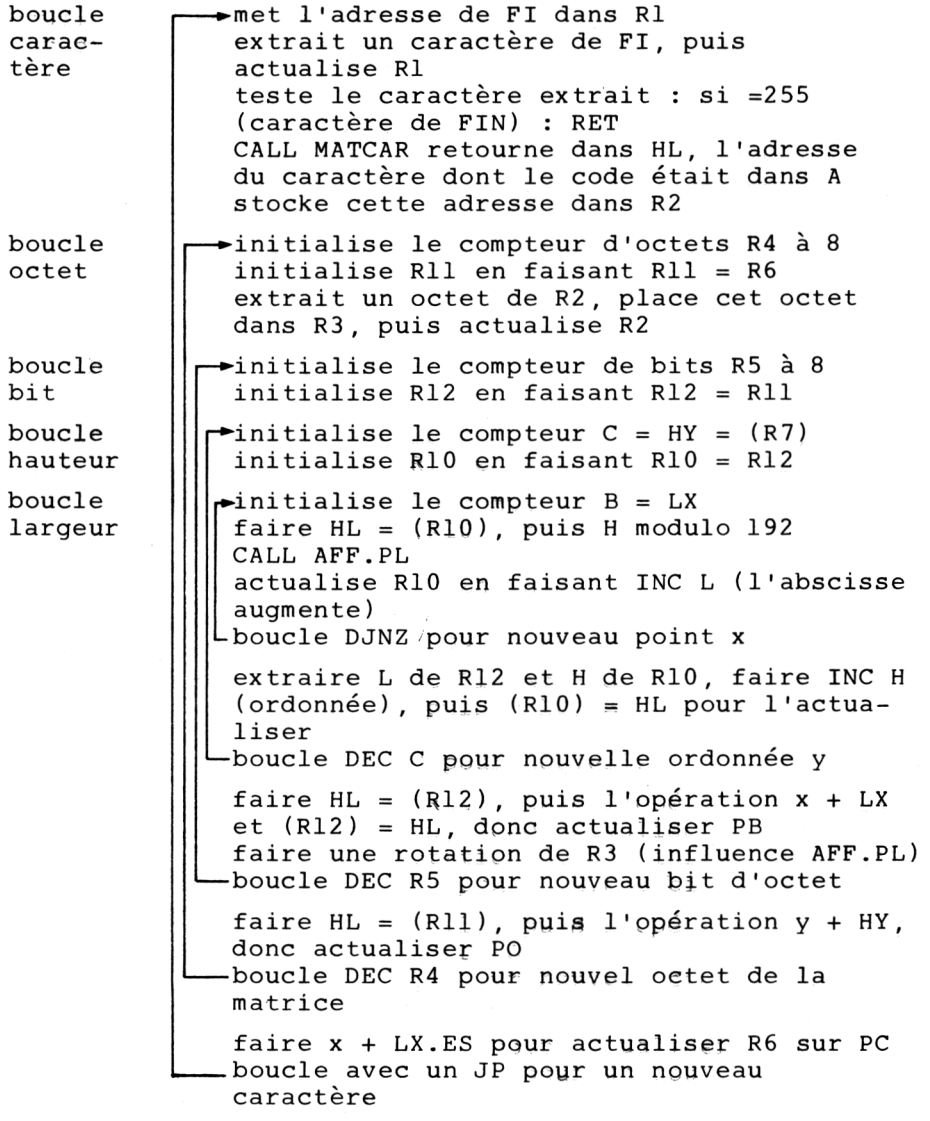
Il est évident, qu'au départ, PA, PB, PO et PC pointent tous sur la position 1 ; puis, PA se déplace dans les 4 positions, puis, PB pointe sur le bit suivant et action de PA, etc..., puis, PO pointe sur l'octet suivant et action de PB et de PA, etc..., puis, PC pointe sur le caractère suivant et action de PO, PB et PA, etc...

Les paramètres et les variables de notre programme nécessitent de nombreux octets de stockage intermédiaire. Nous utilisons, à cette fin, le fichier imprimante qui est inemployé quand on affiche sur l'écran. Voici leurs adresses, leurs noms et leurs usages :

- 23296-7 : R1 : contient l'adresse du caractère à afficher ;
- 23298-9 : R2 : adresse d'un octet de la matrice de caractère ;
- 23300 : R3 : contient un octet de la matrice de caractère ;
- 23301 : R4 : compteur d'octets ;
- 23302 : R5 : compteur de bits ;
- 23303-4 : R6 : PC - position PLOT de départ (x,y) avec x de 0 à 255 dans 23303 et y de 0 à 191 dans 23304 ;
- 23305 : R7 : contient la hauteur HY ;
- 23306 : R8 : contient la largeur LX ;
- 23307 : R9 : contient l'espace ES entre deux caractères ;
- 23308-9 : R10 : position PLOT PA ;
- 23310-11 : R11 : position PLOT PO ;
- 23312-13 : R12 : position PLOT PB ;
- 23320 etc.: FI : codes des caractères à afficher, le dernier doit être 255d.

L'organigramme

Il est difficile de faire un organigramme conventionnel car le texte de remplissage est plutôt dense. L'organigramme, suggéré ci-dessous, contient 5 boucles. L'initialisation se fait dans un programme Basic dans lequel on charge R6, R7, R8, R9 et FI.



Le programme d'impression agrandie

```
            B.CAR
32326 21,18,5B      LD HL,23320    : FI
32329 22,00,5B      LD (23296),HL  : (R1) = FI
32332 2A,00,5B  ┌─→LD HL,(23296)  : HL = (R1)
32335 7E        │   LD A,(HL)      : extrait un
                │                    caractère
32336 23        │   INC HL
32337 22,00,5B  │   LD (23296),HL  : actualise R1
32340 FE,FF     │   CP FF          : est-ce FIN ?
32342 C8        │   RET Z          : si oui, retour
32343 CD,12,7E  │   CALL MATCAR
32346 22,02,5B  │   LD (23298),HL  : (R2) = adresse
                │                    matrice
32349 3E,08  B.OC   LD A,8
32351 32,05,5B  │   LD (23301),A   : (R4) = A
32354 2A,07,5B  │   LD HL,(23303)  : HL = (R6)
32357 22,0E,5B  │   LD (23310),HL  : (R11) = (R6)
32360 2A,02,5B  ├─→LD HL,(23298)  : HL = (R2)
32363 7E        │   LD A,(HL)      : extrait un octet
32364 32,04,5B  │   LD (23300),A   : (R3) = octet
                │                    matrice
32367 23        │   INC HL
32368 22,02,5B  │   LD (23298),HL  : actualise R2
32371 3E,08  B.BIT  LD A,8
32373 32,06,5B      LD (23302),A   : (R5) = A
32376 2A,0E,5B      LD HL,(23310)  : HL = (R11)
32379 22,10,5B      LD (23312),HL  : (R12) = (R11)
            B.HY
32382 3A,09,5B  ┌─→LD A,(23305)  : A = (R7)
32385 4F        │   LD C,A         : C = HY
32386 2A,10,5B  │   LD HL,(23312)  : HL = (R12)
32389 22,0C,5B      LD (23308),HL  : (R10) = (R12)
            B.LX
32392 3A,0A,5B  ┌─→LD A,(23306)  : A = (R8)
32395 47        │   LD B,A         : B = LX
32396 C5        │ ┌→PUSH BC
32397 2A,0C,5B  │ │ LD HL,(23308)  : HL = (R10)
32400 7C        │ │ LD A,H
32401 FE,C0     │ │ CP 192
32403 38,01     │ ┌─JR C, 1+2      : modulo 192
32405 AF        │ │ XOR A
32406 67        │ └→LD H,A
32407 32,0D,5B  │ │ LD (23309),A
32410 CD,2A,7E  │ │ CALL AFF.PL
32413 21,0C,5B  │ │ LD HL,23308    : abscisse
32416 34        │ │ INC (HL)       : +1
32417 C1        │ │ POP BC
32418 10,E8     │ └─DJNZ -24+2
```

```
32420 3A,10,5B  LD A,(23312)
32423 32,0C,5B  LD (23308),A
32426 21,0D,5B  LD HL,23309
32429 34        INC (HL)
32430 0D        DEC C
32431 20,D7     JR NZ, -41+2   : nouvelle ordon-
                                 née y
32433 2A,10,5B  LD HL,(23312)  : HL = (R12)
32436 3A,0A,5B  LD A,(23306)   : A = (R8)
32439 85        ADD A,L        : x + LX
32440 6F        LD L,A
32441 22,10,5B  LD (23312),HL  : (R12) = HL
32444 3A,04,5B  LD A,(23300)   : A = (R3)
32447 17        RLA            : rotation pour
                                 nouveau bit
32448 32,04,5B  LD (23300),A   : (R3) = A
32451 3A,06,5B  LD A,(23302)   : A = (R5)
32454 3D        DEC A
32455 32,06,5B  LD (23302),A   : (R5) = A
32458 20,B2     JR NZ, -78+2
32460 2A,0E,5B  LD HL,(23310)  : HL = (R11)
32463 3A,09,5B  LD A,(23305)   : A = (R7)
32466 84        ADD A,H        : y + HY
32467 67        LD H,A
32468 22,0E,5B  LD (23310),HL  : (R11) = HL
32471 3A,05,5B  LD A,(23301
32474 3D        DEC A
32475 32,05,5B  LD (23301),A
32478 20,88     JR NZ, -120+2
32480 3A,0B,5B  LD A,(23307)   : A = ES
32483 4F        LD C,A
32484 3A,0A,5B  LD A,(23306)   : A = LX
32487 47        LD B,A
32488 3A,07,5B  LD A,(23303)   : A = x
32491 81        ADD A,C
32492 10,FD     DJNZ -3+2      : A = x + LX.ES
32494 32,07,5B  LD (23303),A
32497 C3,4C,7E  JP 32332
```

Nous avons rendu y modulo 192, juste avant l'affichage d'un pixel, car il est difficile de prévoir, vu l'agrandissement, où certains pixels seront affichés. Au delà du fichier d'affichage, se trouvent les attributs qu'il ne faut pas perturber. Notez que, pour x, modulo 256 est automatique et, si vous dépassez à droite, vous afficherez à gauche. Notez également que les registres doubles R6, R10, R11 et R12, contiennent chacun x dans un octet et y dans l'autre ; quelquefois, on les manipule ensemble, quelquefois séparément.

Le lecteur trouvera, ci-dessous, uns transposition Basic qui complète les explications précédentes sur le fonctionnement du programme :

```
 5 BORDER 0
10 LET a$ = "POURQUOI TRAVAILLER EN CODES MACHINE"
15 POKE 32314,32
20 DATA 0,0,1,1,7
30 GO SUB 900
40 LET a$ = "PSI"
50 DATA 0,10,2,2,50
60 GO SUB 900
70 DATA 0,26,3,4,19
80 GO SUB 900
90 DATA 0,52,4,6,8
100 GO SUB 900
105 POKE 32314,40
110 DATA 0,86,5,8,8
120 GO SUB 900
130 DATA 0,128,6,10,8
140 GO SUB 900
145 PAUSE 0
899 STOP
900 FOR n=1 TO 5
910 READ a:POKE 23302+n,a
920 NEXT n
930 LET x=LEN a$
940 FOR n=1 TO x
950 POKE 23319+n,CODE a$(n)
960 NEXT n
970 POKE 23319+n,255
980 RANDOMIZE USR 32326
990 RETURN
```

DATA a été employé pour les 5 paramètres et a$ pour les caractères à traiter. On a fait différents agrandissements : 2.2, 3.4, 4.6, 5.8, 6.10 et, pour la première ligne, où on a mis ES à 7, on a 36 caractères sur une seule ligne. Si vous refaîtes un jeu de caractères avec des matrices 6.8, vous pourrez mettre 42 caractères sur une ligne. Deux lignes sont en inverse, grâce à POKE 32314,40 qui transforme, dans la routine AFF.PL, le JR NZ en JR Z.

Si vous désirez ajouter de la couleur, charger la routine de l'exemple 4 du chapitre III. C'est une coïncidence si cette routine ne change pas d'adresse. Ceci étant fait, ajoutez les lignes Basic suivantes (par exemple) :

```
150 POKE 32500,6 : POKE 32504,88 : POKE 32521,89
151 RANDOMIZE USR 32502
160 POKE 32500,4 : POKE 32504,89 : POKE 32521,90
161 RANDOMIZE USR 32502
170 POKE 32500,5 : POKE 32504,90 : POKE 32521,91
171 RANDOMIZE 32502
175 PAUSE 0
```

CORRIGÉS DES EXERCICES

EXERCICE 1 - Défilement, de haut en bas, d'une ligne de pixels.

```
32500 3E,50   D.BAS LD A,80        : octet haut =
                                     20480
32502 01,E0,07      LD BC,07E0h    : B=7 et C=224
32505 5F            LD E,A         :┐sauve A
32506 80            ADD A,B        :│
32507 57            LD D,A         : │D=A+B
32508 7B            LD A,E         : │restaure A
32509 59            LD E,C         :┘E=C, DE pointe
32510 05            DEC B          : correspond à -256
32511 CB,78         BIT 7,B        : Z = B7
32513 28,0C         JR Z, +12+2    : saut si B> = 0
32515 06,07         LD B,7         : remet B à 7
32517 6F            LD L,A         :┐sauve A
32518 79            LD A,C         :│
32519 D6,20         SUB A,32       : │influence Ci
32521 4F            LD C,A         : │C = C - 32
32522 7D            LD A,L         :┘restaure A
32523 30,02         JR NC, +2+2
32525 D6,08         SUB A,8        : correspond
                                     à -2048
32527 6F            LD L,A         : sauve A
32528 80            ADD A,B        :│
32529 67            LD H,A         : │H=A+B
32530 7D            LD A,L         : │restaure A
32531 69            LD L,C         :┘L=C, HL pointe
32532 C5            PUSH BC        :┐sauve BC
32533 01,20,00      LD BC,32       :│
32536 ED,B0         LDIR           : │transfère une
                                     │ligne de pixels
32538 C1            POP BC         :┘restaure BC
32539 FE,38         CP 56
32541 20,DA         JR NZ,-38+2
32543 C9            RET
```

RANDOMIZE USR 32500 défile d'une ligne de pixels.

Comme pour l'exemple 3, ajoutons :

```
32543 21,00,40      LD HL,16384
32546 06,20         LD B,32
32548 36,00       ┌►LD (HL),0
32550 23          │ INC HL
32551 10,FB       └─DJNZ -5+2
32553 C9            RET
32554 06,80         LD B,128
32556 C5          ┌►PUSH BC
32557 CD,F4,7E    │ CALL D.BAS
32560 C1          │ POP BC
32561 10,F9       └─DJNZ -7+2
32563 C9            RET
```

Même Basic que dans l'exemple précédent. Cependant, si vous désirez voir les 32 derniers octets de la MEM que l'on transfère également, mettez momentanément 201 dans 32543.

CHAPITRE VI

L'ANIMATION

Le chapitre précédent a mis l'accent sur le clavier et l'affichage ainsi que sur les méthodes qui permettent leur mise en oeuvre commune. Le présent chapitre va s'attacher à introduire la dimension du mouvement, essentielle dans les applications ludiques pour lesquelles le ZX-Spectrum est un ordinateur particulièrement désigné.

LES ELEMENTS DE L'ANIMATION : HASARD ET DEPLACEMENT

La simulation de la vie et la création des effets de surprise ne sont pas réalisables sans la prise en compte, par les programmes, de routines de générations aléatoires; quant aux déplacements, ils doivent être décomposés de manière à obtenir le plus de réalisme et de rapidité possibles.

La génération de nombres aléatoires

Il existe plusieurs procédés capables de générer des nombres pseudo-aléatoires plus rapidement que ne le fait la fonction Basic **RND.**

Premier procédé (frames)

Une routine d'interruption intervient toutes les 20 msec. qui, entre autres, actualise un compteur d'images. Ce compteur, qui s'étend sur 3 octets, est une variable système appelée *FRAMES* dont l'octet bas est logé à l'adresse 23672-5C78h. La saisie du contenu donne un nombre entre 0 et 255.

```
3A,78,5C        LD A,(23672)
```

Ce procédé n'est bon que pour saisir un nombre de temps en temps, puisqu'il s'agit d'un compteur fonction du temps qui s'écoule.

Deuxième procédé (MEM)

Dans le même esprit, on peut extraire un octet de la MEM et avoir, ainsi, un nombre également compris entre 0 et 255.

```
32400 21,81,5C  LD HL,23681     : var.sys.inutilisée
32403 34        INC (HL)
32404 6E        LD L,(HL)
32405 26,19     LD H,25         : 6400d comme octet
                                  haut
32407 4E        LD C,(HL)
32 06,00        LD B,O
32410 C9        RET

10 LET a=USR 32400 : PRINT a
20 GO TO 10
```

Troisième procédé (SEED)

Choisissons maintenant un nombre source que l'on stocke dans la variable système SEED : on multiplie ensuite ce nombre par un nombre premier. Le résultat, plusieurs fois modulo 65536, est restocké dans SEED pour usage ultérieur.

Ainsi, par exemple :

```
32400 2A,76,5C ALEA  LD HL,(23670) :: (SEED)
32403 54             LD D,H
32404 5D             LD E,L
32405 29             ADD HL,HL      :  2 fois
32406 19             ADD HL,DE      :  3 fois
32407 29             ADD HL,HL      :  6 fois
```

```
32408 19            ADD HL,DE     :  7 fois
32409 29            ADD HL,HL     : 14 fois
32410 29            ADD HL,HL     : 28 fois
32411 19            ADD HL,DE     : 29 fois
32412 2C            INC L
32413 22,76,5C      LD (23670),HL
32416 44            LD B,H
32417 4D            LD C,L
32418 C9            RET
```

```
10 PRINT USR 32400
20 GO TO 10
```

Au retour de cette routine, que nous appelons ALEA, on prend HL pour avoir un nombre modulo 65536, H ou L pour un nombre modulo 256.

La décomposition des mouvements

En Basic, le temps entre l'effacement de l'ancienne position et l'affichage de la nouvelle doit être le plus court possible pour ne pas laisser l'écran vide trop longtemps. Le calcul de la nouvelle position, ainsi que les tests de rencontre d'obstacles ou des bords de l'écran, s'effectuent donc avant l'effacement. En C.M., nous sommes souvent obligés d'incorporer un retard dans la boucle de fonctionnement pour ralentir le déroulement qui peut être trop rapide.

Voyons comment déplacer un objet, octet par octet, ou ligne pixels par ligne pixels.

Après avoir dessiné l'objet à déplacer, nous l'introduisons dans UDG de 32600 à 32607 et, ensuite :

- initialisation de H (n° de ligne pixels) et de L (n° de colonne)
- détermination du point d'arrivée dans C
- boucle C
  - incrémentation de H
  - faire H + 8
  - placer 32607 dans un DEF
  - faire B = 8
  - boucle B
    - appel de POS.OC
    - transfert de (DEF) vers (DE) : soit de UDG vers l'écran ou affichage
    - actualiser DEF et DE

- DJNZ boucle 8 fois pour 8 octets
- décrémentation de C
- JR NZ, boucle C fois

- retour

Nous supposons que vous avez sauvé les exemples des chapitres précédents ou, tout au moins, les routines principales. Laissez ou chargez les routines suivantes :

```
32200 POS.PR
32240 POS.OC
32258 POS.PL
32274 MATCAR
32298 AFF.PL
```

Entrez, ensuite, le programme :

```
32326 00,00          DEF STO       : stockage intermé-
                                     diaire
32328 26,07  MOUV    LD H,7        : départ -1 du n°
                                     de ligne
32330 2E,1E          LD L,30       : n° de colonne
32332 OE,AA          LD C,170      : arrivée plus 8

32334 24          ┌─►INC H         : donc départ à 8
32335 3E,08       │  LD A,8
32337 84          │  ADD A,H
32338 67          │  LD H,A       : H + 8
32339 11,5F,7F    │  LD DE,32607  : UDG + 7
32342 ED,53,46,7E │  LD (STO),DE  : le stocke dans 32326
32346 06,08       │  LD B,8
32348 C5          │ ►PUSH BC      : sauve B
32349 CD,DO,7D    │  CALL POS.OC  : 32240
32352 00,00,00    │  NOP          : 3 NOP
32355 ED,4B,46,7E │  LD BC,(STO)  : UDG + 7
32359 0A          │  LD A,(BC)
32560 12          │  LD (DE),A
32361 0B          │  DEC BC       : le décrémente
32362 ED,43,46,7E │  LD (STO),BC  : le stocke
32366 25          │  DEC H
32367 C1          │  POP BC       : restaure B
32368 10,EA       └─ DJNZ - 22 + 2 : bouclage de 8 oct.
32370 00,00,00       NOP          : 3 NOP
32373 0D             DEC C
32374 20,D6       ┌─ JR NZ,-42+2  : bouclage C fois
32376 C9          │  RET
```

```
RANDOMIZE USR 32328
```

et voyez comme le A tombe très vite (à l'initialisation du Spectrum, il y a A dans UDG de 32600 à 32607).

Intercalons un retard.

Ajoutons :

```
32370 CD,79,7E      CALL RETARD : 32377
```

en faisant les commandes :

```
POKE 32370,205
POKE 32371,121
POKE 32372,126
```

puis, ajoutons les C.M. du retard :

```
32377 11,01,08   RETARD   LD DE,2049
32380 1B                  DEC DE
32381 7A                  LD A,D
32382 B2                  OR E
32383 20,FB               JR NZ, -5+2
32385 C9                  RET
```

```
RANDOMIZE USR 32328
```

et voyez que le A descend lentement.

Faisons varier sa descente

```
  5 FOR n=8 TO 0 STEP -1
 10 POKE 32379,n
 15 RANDOMIZE USR 32328
 20 NEXT n
999 STOP
```

et voyez les 8 descentes de plus en plus rapides.

Pour montrer l'action du 1er octet de UDG, faisons la commande :

```
POKE 32600,129
```

et vous verrez A descendre comme dans une piste de ski. L'effacement ne se fait plus, il est remplacé par l'affichage de deux points (bit 7 et bit 0).

CONSTRUCTION D'UN JEU D'ANIMATION

Des envahisseurs attaquent

Dans un ciel nocturne étoilé, des envahisseurs vous bombardent. Votre cité, figurée en jaune, est protégée par un dôme rouge. Quand les torpilles qui tombent ont touché le dôme deux fois au même endroit, celui-ci est percé et votre cité détruite. Au premier impact, on remplace le carré rouge par un vert. Heureusement, sur le dôme, vous disposez d'un engin que vous déplacez avec les touches 5 et 8, qui intercepte les torpilles et les détruit.

Reprenons les routines précédentes :

```
32200 POS.PR
32240 POS.OC
32258 POS.PL
32274 MATCAR
32298 AFF.PL
32326 MOUV
```

cette dernière avec des NOP de 32352 à 354 et de 32370 à 372.

Pour dessiner l'endroit d'où partent les torpilles, le dôme et la cité, introduisons une routine pour attribuer des couleurs papier et encre dans les attributs.

```
32377 06,20  COUL  LD B,32
32379 7E           LD A,(HL)
32380 E6,CO        AND CO        : masque papier et
                                   encre
32382 Bl           OR C          : nouvelles couleurs
                                   papier et encre
32383 77           LD (HL),A
32384 23           INC HL
32385 10,F8        DJNZ - 8 + 2
32387 C9           RET
32388 21,CO,5A ENT LD HL,23232   : ligne 22
32391 OE,10        LD C,16       : papier rouge,
                                   encre noire
32393 CD,79,7E     CALL COUL
32396 21,EO,5A     LD HL,23264   : ligne 23
32399 OE,30        LD C,48       : papier jaune,
                                   encre noire
```

```
32401 CD,79,7E      CALL COUL
32404 21,00,58      LD HL,22528 : ligne 0
32407 OE,00         LD C,O      : papier et encre
                                  noirs
32409 CD,79,7E      CALL COUL
32412 C9            RET

10 RANDOMIZE USR 32388
20 PAUSE 0
```

Cette ligne Basic est nécessaire pour voir les lignes 22 et 23.

Construisons également deux routines pour déplacer notre engin à gauche et à droite.

```
32413 11,20,01 DE.G LD DE,288    : 256 + 32
32416 21,BF,50      LD HL,20671 : fin ligne 21
32419               LD C,8
32421 06,20       ┌►LD B,32
32423 A7          │ AND A        : ci = 0
32424 7E          │►LD A,(HL)    : saisie d'un octet
32425 17          │ RLA          : rotation 9 bits
32426 77          │ LD (HL),A
32427 2B          │ DEC HL       : octet précédent
32428 10.FA       └─DJNZ - 6 + 2 : boucle les 32 octets
32430 19          ┌─ADD HL,DE
32431 OD          │ DEC C
32432 20,F3       └─JR NZ,- 13 + 2 : boucle les 8 li-
                                     gnes pixels (de
32434 C9            RET              la ligne 21)
```

et pour le déplacement vers la droite :

```
32435 11,E0,00 DE.D LD DE,224    : 256 - 32
32438 21,AO,50      LD HL,20640  : début ligne 21
32441 OE,08         LD C,8
32443 06,20       ┌►LD B,32
32445 A7          │ AND A
32446 7E          │►LD A,(HL)
32447 1F          │ RRA
32448 77          │ LD (HL),A
32449 23          │ INC HL
32450 10,FA       └─DJNZ - 6 + 2
32452 19          ┌─ADD HL,DE
32453 OD          │ DEC C
32454 20,F3       └─JR NZ, - 13 + 2
32456 C9            RET
```

Ajoutons une routine pour afficher l'engin,

```
32457 11,AF,50 AFF LD DE,20655  : ligne 21, colonne 16
32460 21,60,7F     LD HL,32608  : UDG 2e caractère
32463 06,08        LD B,8       :
32465 7E        ┌─►LD A,(HL)
32466 00        │  NOP
32467 00        │  NOP
32468 12        │  LD (DE),A
32469 14        │  IND D       : ajoute 256
32470 23        │  INC HL
32471 10,F8     └──DJNZ - 8 + 2
32473 C9           RET
```

Pour que les torpilles ne tombent pas toujours du même endroit, faisons une routine **ALEA** adaptée de celle du début de ce chapitre.

```
32474 2A,76,5C ALEA LD HL,(23670): (SEED)
32477 54           LD D,H
32478 5D           LD E,L
32479 29           ADD HL,HL
32480 19           ADD HL,DE
32481 29           ADD HL,HL
32482 19           ADD HL,DE
32483 29           ADD HL,HL
32484 29           ADD,HL,HL
32485 19           ADD HL,DE
32486 2C           INC L
32487 22,76,5C     LD (23670),HL
32490 7C           LD A,H
32491 E6,1F        AND 31
32493 6F           LD L,A       : L modulo 32
32494 C9           RET
```

Nous avons terminé les routines dont nous aurons besoin. Avant d'aborder le jeu proprement dit, il faut ajouter une ligne dans la routine **MOUV** :

```
32352 C3,49,7D     JP TEST      : 32073
```

en faisant
```
POKE 32352,195
POKE 32353,73
POKE 32354,125
```

L'organisation du jeu

Nous chargeons le jeu à 32000 (599 octets avec le programme de chargement). Les routines restent à 32200 et seq. Chaque fois que vous sauvez, faîtes : SAVE "CM" CODE 32000,767.

```
32000 00,00          DEF compteur
32002 00,00          DEF mouv.
          JEU
32004 21,00,00       LD HL,0
32007 22,00,7D       LD (32000),HL : RAZ le compteur
32010 CD,84,7E       CALL ENT      : 32388
32013 CD,C9,7E       CALL AFF      : 32457
32016 CD,DA,7E       CALL ALEA     : 32474
32019 26,08          LD H,8        : ne pas dépasser 183
32021 22,02,7D       LD (mouv),HL
        BOUCLE                     : reprend la ligne
32024 2A,02,7D       LD HL,(mouv)    pixels
32027 24             INC H         : augmente la ligne
                                     pixels
32028 22,02,7D       LD (mouv),HL  : la stocke
32031 0E,01          LD C,1        : 1 pixel
32033 CD,4E,7E       CALL MOUV     : 32334 autre entrée
          TOUCHE
32036 CD,8E,02       CALL SCRUT    : 654d
32039 7B             LD A,E
32040 FE,13          CP 19         : touche 8
32042 20,05          JR NZ,+5+2
32044 CD,B3,7E       CALL DE.D
32047 18,0D          JR +13+2
32049 FE,04          CP 4          : touche 5
32051 20,05          JR NZ,+5+2
32053 CD,9D,7E       CALL DE.G
32056 18,04          JR +4+2
32058 06,10 RETSUP   LD B,16       : retard supplément.
32060 18,02          JR +2+2
32062 06,0D RETARD   LD B,13
32064 0E,FF          LD C,255
32066 0D             DEC C
32067 20,FD          JR NZ,-3+2
32069 10,F9          DJNZ -7+2
32071 18,CF          JR BOUCLE-49+2
```

Le jeu est ainsi construit :

- appel routine **ENT** pour afficher 3 bandes de couleur ;
- appel routine **AFF** pour afficher l'engin ;
- appel routine **ALEA** pour que les torpilles partent d'un endroit de la voûte au hasard.
- BOUCLE appel **MOUV** pour déplacer la torpille - voir TESTS ;
  appel **SCRUT** pour déplacer l'engin avec les touches 5 et 8 (explications dans le chapitre VI).

En absence de déplacement, augmenter le retard :

```
RETARD
bouclage
```

Nous continuons avec TESTS qui intervient dans la routine **MOUV** à la ligne 32352 :

```
32073 1A  TESTS    LD A,(DE)     : DE=pos. octet
32074 FE,66        CP 102        : engin
32076 20,0B        JR NZ,+11+2
32078 ED,4B,00,7D  LD BC,(compt)
32082 03           INC BC        : augmente le
                                   compteur
32083 ED,43?00,7D  LD (compt),BC: le stocke
32087 18,0A        JR +10+2
32089 0A           LD A,(BC)     : BC=pos. attribut
32090 E6,38        AND 56        : isole la couleur
32092 FE,10        CP 16         : est-ce rouge ?
32094 20,1B        JR NZ,+27+2
32096 3E,20        LD A,32       : couleur verte
32098 02           LD (BC),A
32099 C1           POP BC        : actualise SP
32100 C1           POP BC        : actualise SP
32101 11,E0,00     LD DE,224     : pour effacer la
                                   ligne 21
32104 21,A0,50     LD HL,20640
32107 0E,08        LD C,8
32109 06,20        LD B,32
32111 36,00        LD (HL),0
32113 23           INC HL
32114 10,FB        DJNZ -5+2
32116 19           ADD HL,DE
32117 0D           DEC C
32118 20,F5        JR NZ,11+2
32120 C3,0D,7D     JP 32013      : nouveau départ
32123 FE,20        CP 32         : est-ce vert ?
```

```
32125 20,07          JR NZ, + 7 + 2
32127 C1             POP BC
32128 C1             POP BC
32129 ED,4B,00,7D    LD BC,(compt)
32133 C9             RET            : retour au Basic
32134 C3,63,7E       JP 32355       : dans MOUV
```

Les tests portent sur la position torpille :

- si torpille = engin
  - . augmenter le score ;
  - . effacer la torpille ;
  - . POP BC deux fois pour actualiser la pile puisqu'on sort de la routine MOUV ;
  - . aller à nouveau départ.
- si torpille = carré rouge
  - . mettre la couleur verte ;
  - . POP BC deux fois ;
  - . effacer la torpille ;
  - . aller à nouveau départ.
- si torpille = carré vert
  - . POP BC deux fois
  - . charger le compteur dans BC avant le RET au Basic ;
  - . RET au Basic
- si torpille = rien
  - . continuer en retournant à 32355 dans la routine MOUV.

Pour terminer, ajoutons l'exemple 9 du chapitre II :

```
32137 21,00,19      LD HL,6400
32140 11,00,58      LD DE,22528
32143 01,00,03      LD BC,768
32146 ED,B0         LDIR
32148 C9            RET
```

Quand vous aurez l'habitude, vous pouvez augmenter la vitesse en diminuant les contenus de 32059 et 32063. Vous pouvez aussi faire démarrer les torpilles plus bas en augmentant le contenu de 32020.

Notez que ce programme, comme la plupart des programmes de ce livre, n'est pas optimisé. Notre but est didactique. Il vise à vous faire appréhender l'emploi progressif des C.M.

VARIATION DE LA VITESSE

Pour déplacer deux objets A et B, à des vitesses différentes, on emploie le procédé suivant :

- 1er passage dans une boucle : mouvement de A
- 2e passage dans une boucle : mouvement de A et mouvement de B
- 3e passage dans une boucle : mouvement de A
- 4e passage dans une boucle : mouvement de A et mouvement de B

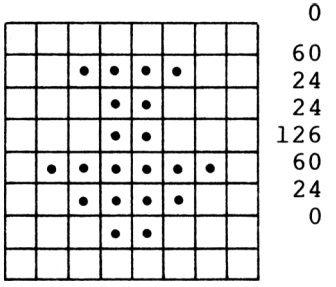
etc.

L'objet A ira deux fois plus vite que l'objet B.

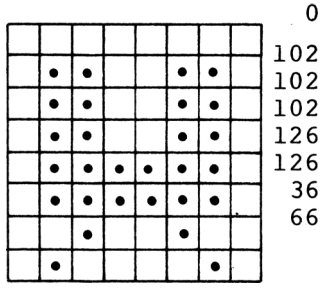
Pour exploiter le jeu, entrons le Basic suivant :

```
 10 BORDER 1: PAPER 1: INK 7: CLS
 20 FOR n=1 TO 20
 30 PRINT AT RND*19+1,RND*31 ; "."
 40 NEXT n
 50 DATA 0,60,24,24,126,60,24,0
 55 DATA 0,102,102,102,126,126,36,66
 60 FOR n=32600 TO 32615
 70 READ x: POKE n,x
 80 NEXT n
 90 LET x=USR 32004
100 PAPER 7: INK 0
110 PRINT AT 0,0 ; "Vous avez détruit" ; x ; "torpilles"
120 FOR n=1 TO 500: NEXT n
130 RANDOMIZE USR 32137
140 PRINT AT 10,9 ; PAPER 7 ; INK 0 ; "C'est le chaos"
150 BEEP .25,7: BEEP .75,7: BEEP .25,7: BEEP 1,12:
    BEEP .75,12: BEEP .75,14: BEEP .75,14: BEEP 1.25,19;
    BEEP .25,16: BEEP 1,12
999 STOP
```

La torpille et l'engin se déplacent à la même vitesse, mais tous deux le font pixel par pixel, donc pour que l'engin intercepte la torpille, il faut qu'il soit juste en dessous : à un pixel près, il n'agit pas car, dans les tests, on compare à 102 qui est le premier octet de l'engin. Les lignes Basic 50 à 80 mettent les octets de la torpille et de l'engin dans UDG. Nous les avons dessiné comme suit :



Torpille



Engin

Pour réaliser ceci, on fait un test au début des routines de mouvement, à l'aide d'un octet mémoire réservé à cet effet.

```
BOUCLE  ┌─►LD A,(nm)      : nm est l'adresse de
        │                   l'octet réservé
        │  INC A
        │  BIT 0,A        : Z=Ā0 : test sur A0
        │ ┌JR NZ,MOUV A
MOUV B  │ │programme du
        │ │mouvement de B
MOUV A  │ └►programme du
        │   mouvement de A
        └──JR BOUCLE
```

Une fois sur deux, JR,NZ, ira à MOUV A puisque A0=1 une fois sur deux.

Pour augmenter la souplesse de ce dispositif, on emploie deux indicateurs pour les deux mouvements :

```
BOUCLE  ┌─►LD B,2
MOUV A  │ ┌►programme du
        │ │ mouvement de A
        │ └─DJNZ
        │   LD C,1
MOUV B  │ ┌►programme du
        │ │ mouvement de B
        │ └─JR NZ,
        └──JR BOUCLE
```

et ainsi, en modifiant B et C à volonté, on arrive à déplacer deux objets à des vitesses variables. Cependant, de grandes différences entre B et C peuvent amener des mouvements saccadés ; il est préférable, alors, d'imbriquer les mouvements. Ce système s'étend facilement à plus de deux objets. L'une des vitesses peut être aléatoire, ce que nous réalisons dans le jeu suivant.

Attaque d'un avion

Un avion survole votre batterie anti-aérienne. Si vous n'avez pas descendu l'avion après 5 passages, celui-ci vous a repéré et vous détruit sans vous rater. A chaque passage, l'avion se trouve à une hauteur différente et sa vitesse est constamment variable.

Comme pour l'exemple précédent, reprenons les routines suivantes :

32200 **POS.PR**
32240 **POS.OC**
32258 **POS.PL**
32274 **MATCAR**
32298 **AFF.CA**

Cette dernière routine n'est pas relogeable telle quelle, sans changer les deux lignes suivantes :

```
32299 CD,C8,7D     CALL POS.PR : changement adresse
32304 CD,12,7E     CALL MATCAR : changement adresse
```

Nous y ajoutons quelques routines adaptées à nos besoins :

32320 **ALEA** : retour L contient un nombre de 1 à 4 et A un nombre de 0 à 7 qu'il faut sauver dans 32319 ;

32349 **COUL** : avant l'appel, on charge HL = adresse du début de ligne ATTR, C = papier * 8 + encre et B = nombre d'octets ATTR à charger ;

32361 **MOUV** : meut une colonne pixel par pixel. Avant l'appel, charger le nombre de boucles à effectuer dans 32358 ;

32414 **DE.G** : défile une ligne de caractères à gauche, pixel par pixel. Avant l'appel, il faut un CALL ALEA pour avoir la hauteur et mettre 0 dans 32412 pour contrôler l'effacement et l'affichage ;

32459 **AVION** : affiche l'avion. Utilise trois caractères UDG et la routine DE.G pour l'afficher

Routine ALEA

```
32319 00       ALEA DEF hauteur
32320 2A,76,5C      LD HL,(23670): (SEED)
32323 54            LD D,H
32324 5D            LD E,L
32325 29            ADD HL,HL
32326 19            ADD HL,DE
32327 29            ADD HL,HL
32328 19            ADD HL,DE
32329 29            ADD HL,HL
32330 29            ADD HL,HL
32331 19            ADD HL,DE
32332 2C            INC L
32333 22,76,5C      LD (23670),HL
32336 7C            LD A,H
32337 E6,03         AND 3         : A = de 0 à 3
32339 3C            INC A
32340 6F            LD L,A        : L = de 1 à 4
32341 7B            LD A,E
32342 E6,07         AND 7         = A = de 0 à 7
32344 C9            RET
```

Routine COUL

```
32345 00            NOP
32346 00            NOP
32347 00            NOP
32348 00            NOP
32349 7E  COUL      LD A,(HL)
32350 E6,C0         AND COh       : masque papier et
                                    encre
32352 B1            OR C          : nouveau papier et
                                    encre
32353 77            LD (HL),A
32354 23            INC HL
32355 10,F8         DJNZ -8+2
32357 C9            RET
```

Routine MOUV

```
32358 00            DEF boucle   : à initialiser
32359 00,00         DEF STO      : stockage inter-
                MOUV               médiaire
32361 3A,3F,7E  ┌──►LD A,(32319): hauteur de 0 à 7
32364 3C        │   INC A        : de 1 à 8
32365 87        │   ADD A,A      : 2 *
32366 87        │   ADD A,A      : 4 *
32367 87        │   ADD A,A      : 8 *
32368 C6,40     │   ADD A,64     : 8*A + 64 = 72 à 128
32370 67        │   LD H,A
32371 2E,0F     │   LD L,15      : colonne 15
32373 CD,FO,7D  │   CALL POS.OC
32376 ED,53,67,7E   LD (DEFSTO),DE
32380 C3,A9,7D  │   JP TEST      : 32169 voir note
32383 24        │ ┌►INC H
32384 CD,F0,7D  │ │ CALL POS.OC
32387 ED,4B,67,7E   LD BC,(DEFSTO)
32391 1A        │ │ LD A,(DE)    : contenu pos. H+1
32392 02        │ │ LD (BC),A    : dans contenu pos. H
32393 ED,53,67,7E   LD (DEFSTO),DE
32397 3E,A0     │ │ LD A,160
32399 BC        │ │ CP H         : ligne pixel 160 ?
32400 20,ED     │ └─JR NZ, - 19 + 2
32402 3A,66,7E  │   LD A,(boucle):(32358)
32405 3D        │   DEC A
32406 32,66,7E  │   LD (boucle),A
32409 20,CE     └──►JR NZ, - 50 + 2
32411 C9            RET
```

A 32380, il faut charger 00,00,00 si vous voulez essayer la routine avant d'avoir charge la routine TEST à 32169.

La boucle de 32383 à 32400 va plus vite quand la hauteur est 7, et de moins en moins vite quand la hauteur tend vers 0.

Routine DE.G

```
32412 00            DEF effa
32413 00            NOP
32414 3A,3F,7E DEG  LD A,(haut.): 0 à 7
32417 87            ADD A,A      : *2
32418 87            ADD A,A      : *4
32419 87            ADD A,A      : *8
32420 87            ADD A,A      : *16
32421 87            ADD A,A      : *32
32422 C6,1F         ADD A,31     : A=31, 63, 95... 255
```

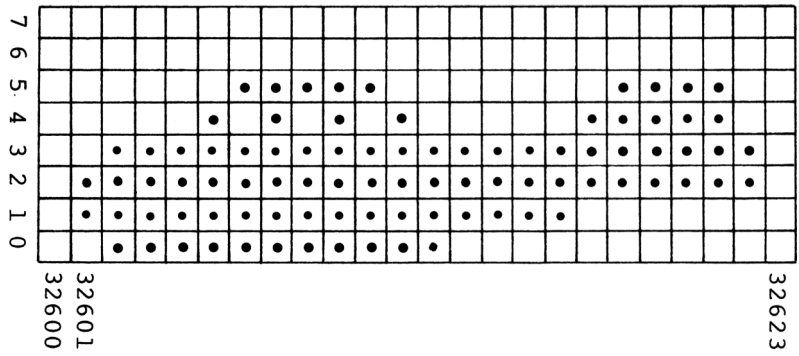
```
32424 6F           LD L,A
32425 26,48        LD H,72      : réalise 18432
                                  comme octet haut
32427 11,20,01     LD DE,288    : 256 + 32
32430 0E,08        LD C,8       : 8 lignes pixels
                                  d'un caractère
32432 3A,9C,7E     LD A,(effa)
32435 17           RLA
32436 32,9C,7E  ┌─►LD (effa),A
32439 06,20     │  LD B,32      : déplace une ligne
                │                 de pixels
32441 7E        │ ┌►LD A,(HL)
32442 17        │ │ RLA
32443 77        │ │ LD (HL),A
32444 2B        │ │ DEC HL
32445 10,FA     │ └─DJNZ - 6 + 2
32447 19        │   ADD HL,DE
32448 0D        │   DEC C
32449 20,ED     └───JR NZ, - 19 + 2
32451 C9            RET
```

Au préalable, ALEA doit être appelée pour que la hauteur soit chargée ; notez que, sans cela, la hauteur est chargée au départ à 0 et la routine n'est pas affectée dans son fonctionnement. Après 32425, HL pointe sur la fin des lignes 8, 9, 10... 15 de position PRINT.

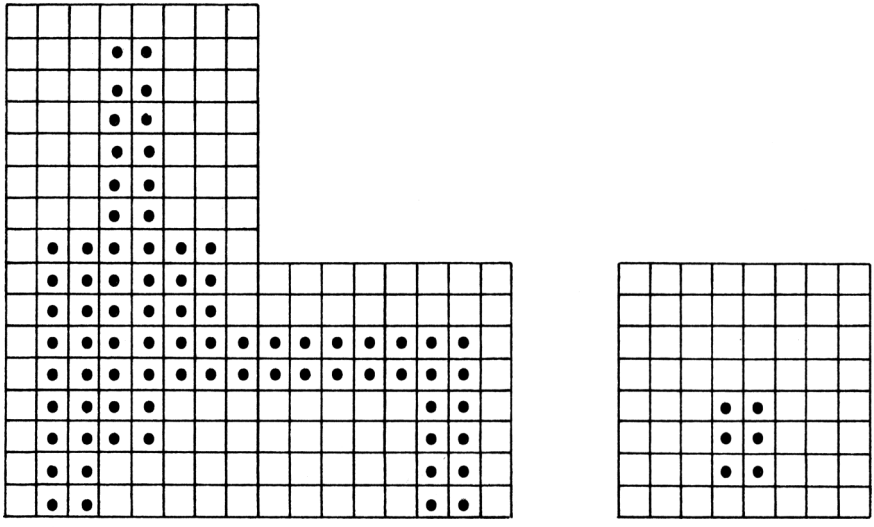
Routine AVION

```
32459 06,18 AVION  LD B,24      : 24 octets de UDG
32461 21,58,7F     LD HL,32600  : début de UDG
32464 C5        ┌─►PUSH BC
32465 E5        │  PUSH HL
32466 7E        │  LD A,(HL)    : 1 octet de UDG
32467 32,9C,7E  │  LD (effa),A  : dans effa
32470 CD,9E,7E  │  CALL DE.G    : 32414
32473 E1        │  POP HL
32474 C1        │  POP BC
32475 23        │  INC HL
32476 10,F2     └──DJNZ - 14 + 2
32478 C9           RET
```

L'avion s'affiche à droite d'une ligne 8 à 15.



Attention : l'affichage de l'avion se fait latéralement ; de ce fait, l'octet frontal apparaît d'abord. Le chargement dans UDG en est affecté : il n'est pas dans l'ordre habituel.



Voici un Basic de chargement possible :

```
900 DATA 0,6,15,15,15,31,47,63
910 DATA 47,63,47,31,15,14,14,14
920 DATA 14,28,60,60,60,60,12,0
930 DATA 0,24,24,24,24,24,24,24
940 DATA 126,126,127,127,120,120,96,96
950 DATA 0,0,254,254,6,6,6,6
```

```
960 DATA 0,0,0,0,24,24,24,0
970 FOR n=32600 TO 32655
980 READ x : POKE n,x : NEXT n
```

Quand vous aurez enregistré les octets situés après RAMTOP en faisant

```
SAVE "CM" CODE 32000,767
```

vous pourrez effacer ce Basic

L'organisation du jeu de l'avion

Les éléments d'organisation du jeu sont les suivants :

- écran cyan ;
- le sol vert ;
- encre cyan sur les 8 premières lignes pour affichage instantané de petits nuages en faisant, plus loin, encre blanche :
- affichage invisible de petits nuages sur les 8 premières lignes, par transfert de la MEM ;
- affichage du canon ;
- initialisation du compteur T : nombre de tours ou passages d'avion : fixé à 5 ;
- boucle 1
  - CALL ALEA et stocker la hauteur ;
  - affichage d'une ligne de petits nuages qui simulent la traînée laissée par un avion ;
  - affichage avion ;
  - initialisation du compteur P : nombre de déplacements de l'avion sur un tour : soit 256 puisqu'il y a 256 pixels en largeur ;
- boucle 2
  - scrutation du clavier et saisie de la touche F : si la touche est pressée, affichage de l'obus ;
  - nombre de déplacements de l'obus : fixé à 4 ;
  - déplacements de l'avion après appel d'ALEA pour fixer les déplacements de 1 à 4 : donc, mouvement saccadé de l'avion pour éviter les obus ;
  - retard de déroulement ;
  - bouclage b2 ;
  - sauf si le compteur P = 256 ;
  - bouclage b1 ;

- sauf si le compteur T = 5, alors RET au Basic pour explosion du canon.

- TEST : dans la routine MOUV, nous renvoyons à ce test qui consiste à savoir si la partie de l'obus 24 (il y en a 3) est à proximité immédiate de la partie de l'avion d'octet 255 (les 8 bits à 1). Si oui, RET au Basic pour explosion de l'avion

En examinant le dessin de l'avion, vous voyez qu'il y a, en dessous, 11 bits à 1. L'octet de comparaison sera donc 4 fois à 255 pendant 4 déplacements, puis celui-ci sera invulnérable. De plus, vous avez 3 octets à 24 dans le dessin de l'obus, vos chances d'abattre l'avion sont grandes. Pour diminuer vos chances, vous pouvez remplacer deux octets de l'obus par 28 au lieu de 24 ou encore en ne laissant que 10, 9 ou 8 bits à 1, sur le dessous de l'avion, en remplaçant, dans l'octet 32602, 15 par 14, idem pour 32603 et 32604.

A ce moment, vous n'aurez plus qu'une chance de coïncidence.

```
32000 00            DEF compteur P
32001 00            DEF compteur T
32002 à 32009 00    NOP
32010 21,00,5A SOL  LD HL,23232  : début ligne 22
                                   de ATTR
32013 0E,20         LD C,32      : papier vert,
                                   encre noire
32015 06,40         LD B,64      : remplit les li-
                                   gnes 22 et 23
32017 CD,5D,7E      CALL COUL    : 32349
32020 21,00,58      LD HL,22528  : début ligne 0
              ENCRE                de ATTR
32023 0E,2D         LD C,45      : papier cyan,
                                   encre cyan
32025 06,00         LD B,256     : lignes 0 à 7
32027 CD,5D,7E      CALL COUL
32030 21,00,40 MEM  LD HL,16384  : début fichier
                                   affichage
32033 11,00,56      LD DE,14336  : pointe dans la MEM
32036 1A          ┌→LD A,(DE)
32037 77          │ LD (HL),A
32038 13          │ INC DE
32039 23          │ INC HL
32040 7C          │ LD A,H
32041 FE,48       │ CP 72
32043 20,F7       └─JR NZ, - 9 + 2
```

```
                CANON
32045 21,0F,14        LD HL,140F    : H=20 et L=15
32048 3E,93           LD A,147      : graphique d
32050 CD,2A,7E        CALL AFF.CA   : 32298
32053 21,0F,15        LD HL,150F    : H=21 et L=15
32056 3E,94           LD A,148      : graphique e
32058 CD,2A,7E        CALL AFF.CA
32061 21,10,15        LD HL,1510h   : H=21 et L=16
32064 3E,95           LD A,149      : graphique f
32066 CD,2A,7E        CALL AFF.CA
32069 AF     INIT.T   XOR A         : A=0
32070 32,01,7D        LD (32001),A  : met T à 0
            BOUCLE 1
32073 CD,40,7E        CALL ALEA     : 32320
32076 32,3F,7E        LD (32319),A  : stocke la hauteur
32079 26,58 NUAGE     LD H,88       : 22528 début fi-
                                      chier ATTR
32081 OF              RRCA          : x32 quand A ne
                                      contient que A0,
                                      A1 et A2
32082 OF              RRCA
32083 OF              RRCA
32084 FF              LD L,A        : ligne 0 à 7
32085 0E,2F           LD C,47       : papier cyan,
                                      encre blanche
32087 06,20           LD B,32       : remplit une ligne
32089 CD,5D,7E        CALL COUL
                AVION
32092 CD,CB,7E        CALL AVION    : 32459
32095 AF     INIT.P   XOR A         : A=0
32096 32,00,7D        LD (32000),A  : met P à 0
            BOUCLE 2
32099 CD,8E,02        CALL SCRUT    : 654d
32102 7B              LD A,E        : saisie d'une touche
32103 FE,0E           CP 14         : est-ce F ? (voir
                                      chapitre VI)
32105 20,08           JR NZ, + 8 + 2
            AF.OBUS
32107 21,0F,13        LD HL,130F    : H=19 et L=15
32110 3E,96           LD A,150      : graphique g
32112 CD,2A,7E        CALL AFF.CA
32115 3E,04 NOMBRE    LD A,4        : 4 boucles de MOUV
32117 32,66,7E        LD (32358),A
                OBUS
32120 CD,69,7E        CALL MOUV
```

```
                AVION
32123 CD,40,7E  |   CALL ALEA
32126 7D        |   LD A,L        : de 1 à 4
32127 47        |   LD B,A
32128 C5        |  >PUSH BC       : sauve B
32129 CD,9E,7E  |   CALL DE.G     : 32414
32132 C1        |   POP BC        : restaure B
32133 3A,00,7D  |   LD A,(compt P)
32136 3C        |   INC A
32137 32,00,7D  |   LD (compt P),A
32140 28,0D     |  -JR Z, + 13 + 2 : saut après 256
                |                   déplacements
32142 10,F0     |  -DJNZ - 16 + 2  : de 1 à 4 dépla-
                |                   cements
32144 06,01 RETARD LD B,1       : retard minimum
32146 0E,01     |  >LD C,1
32148 0D        |  >DEC C
32149 20,FD     |  -JR NZ, - 3 + 2
32151 10,F9     |  -DJNZ - 7 + 2
32153 18,C8     |  -JR - 56 + 2    : BOUCLE 2
32155 3A,01,7D  |  >LD A,(compt T)
32158 3C        |   INC A
32159 FE,05     |   CP 5
32161 32,01,7D  |   LD (compt T),A
32164 28,02     |  -JR Z, + 2 + 2  : saut après 5 tours
32166 18,A1     |  -JR - 95 + 2    : BOUCLE 1
32168 C9        |  >RET            : au Basic : exp.can
32169 1A      TEST  LD A,(DE)
32170 FE,18     |   CP 24
32172 20,09     |  -JR NZ, + 9 + 2 : saut si pas d'obus
32174 25        |   DEC H
32175 CD,F0,7D  |   CALL POS.OC    : pos. ligne pixels
                |                   précédente
32178 1A        |   LD A,(DE)
32179 3C        |   INC A
32180 28,04     |  -JR Z, + 4 + 2  : saut si avion
                |                   (255+1)
32182 24        |   INC H
32183 C3,7F,7E  |  >JP 32383
32186 C1        |  >POP BC         : actualise SP
32187 C9        |   RET            : au Basic, expl.
                |                   avion
```

Après avoir entré la routine **TEST**, vous devez modifier MOUV :

```
32380 C3,A9,7D     JP 32169
```

en faisant
```
POKE 32380,195
POKE 32381,169
POKE 32382,125
```

Avant le retour au Basic, il faut actualiser SP car le saut à 32169 vient de la routine **MOUV** qui a été appelée. Sur cet appel, le microprocesseur pousse l'adresse de retour dans SP. Comme on a appelé MOUV à 32120, l'adresse de retour est 32123 ; de ce fait, cette adresse se trouve dans BC avant le RET. Nous utilisons cette adresse dans le Basic d'exploitation qui suit pour savoir d'où provient le retour au Basic : soit de 32168, soit de 32187.

```
 10 BORDER 0 : PAPER 5 : INK 0 : CLS
 20 LET bc=USR 32010
 30 IF bc=32123 THEN GO TO 100
 35 FOR n=0 TO 7
 40 PRINT AT 21,14 ; PAPER n ; INK 7-n ; "***"
 45 PAUSE 20 : NEXT n
 50 BEEP 1,0 : BEEP .75.0 : BEEP .250 : BEEP 1,0 : BEEP 1,3 :
    BEEP .25,2 : BEEP .75,2 : BEEP .25,0 : BEEP .75,0 :
    BEEP .25,-1 : BEEP 1,0
 90 CLS : PRINT AT 11,8 ; "Vous avez perdu"
 99 STOP
100 LET 1=PEEK 32319
110 LET 1=1+8
120 FOR n=0 TO ?
130 PRINT AT 1,14 ; PAPER n ; INK 7-n ; "****"
140 PAUSE 20 : NEXT n
145 FOR n=1 TO 3
150 BEEP 1,12 : BEEP .25,7 : BEEP .25,9 : BEEP .25,7 :
    PAUSE 40
155 NEXT n
190 CLS : PRINT AT 11,5 ; "Bravo, vous l'avez eu"
999 STOP
```

Poussez sur F pour envoyer des obus.

CHAPITRE VII

LE MONITEUR

Dans les pages précédentes, il a été souvent fait appel à des routines du programme moniteur (MEM). Le moment est venu de présenter celui-ci d'une manière plus systématique, en signalant les principaux points d'entrée et en démontant les routines les plus intéressantes à intégrer dans des programmations personnels. Ce chapitre se terminera par une illustration graphique mettant en oeuvre les routines complexes de calcul en virgule flottante

LES POINTS D'ENTREE

Certaines instructions du Z80A sont spécialement destinées à appeler des routines de la MEM ; elles figurent au début du programme moniteur ("page zéro" ; elles ont, pour mnémonique, RST.

Les appels de sous-programme en page zéro (RST)

Voici les 8 mnémoniques RST avec leurs C.M. :

C7	RST 0	q=00
CF	RST 8	q=08
D7	RST 10	q=16d
DF	RST 18	q=24d
E7	RST 20	q=32d
EF	RST 28	q=40d
F7	RST 30	q=48d
FF	RST 38	q=56d

Chaque C.M. exécute :

```
(SP-1) = PChaut
(SP-2) = PCbas
SP     = SP-2
PChaut = 0
PCbas  = q
```

D'après les opérations effectuées par un RST, vous voyez que ce mnémonique travaille comme un CALL, mais ne nécessite qu'un seul C.M.. Ces instructions sont au début de la MEM et ne comportent que 8 octets de C.M., à l'exception de la 8ème adresse qui commence à 56d et qui peut comporter autant d'octets que l'on veut. Cette faculté, offerte par le Z80A, est exploitée par les programmeurs de MEM, selon leurs conceptions : elles correspondent, dans le ZX-Spectrum, a un emploi particulier à cet ordinateur.

RST 0

Ce C.M. n'est pas employé dans un programme : il s'exécute automatiquement à la mise sous tension.

C'est le RESET du microprocesseur qui procède à l'initialisation du système : A est mis à 0, C_i à 0 et DE à 65535, puis l'ordinateur saute à 4555 qui est le début de la routine d'initialisation (qui se termine à 4769). Cette routine calcule la mémoire disponible, remplit les variables système adéquates et rend les interruptions possibles avec EI (voir plus loin) ; elle se termine par l'édition du message "copyright" ; ensuite, l'ordinateur entre dans une boucle d'attente.

RST 8

C'est le routine des messages de compte-rendu. RST 8 est suivi d'un octet de DEF qui est copié dans la variable système 23610 et qui détermine le code d'erreur employé. L'emploi de RST 8 provoque l'arrêt immédiat.

Faîtes, par exemple :

```
POKE 28000,207     pour CF      RST 8
POKE 28001,28      pour 1C      DEF T
```

puis **RANDOMIZE USR** 28000, avec un message que vous connaissez ; si vous essayez POKE 28001,27 pour DEFS, vous aurez un message inattendu.

Si vous chargez un nombre inférieur ou égal à 26, vous aurez les messages qui accompagnent R et en dessous ; il faut charger 255 pour avoir le report 0 OK.

RST 10

C'est la routine d'impression d'un caractère dont le code se trouve dans A. Par exemple, introduire :

```
28000 3E,41        LD A,65       : code de A
                                   POKE 28000,62
                                   POKE 28001,65
28002 D7           RST 10        : POKE 28002,215
28003 C9           RET           : POKE 28003,201
```

puis RANDOMIZE USR 28000 ; et ... vous ne voyez rien. La position PRINT se trouve dans la partie inférieure de l'écran, et le compte-rendu surcharge le A.

Introduisez alors en Basic :

```
10 PRINT AT 0,0 ; " " ;
20 RANDOMIZE USR 28000
```

et vous avez A au dessus, à gauche à (0,1).

Il est donc nécessaire de déterminer la position **PRINT** avant d'appeler la routine située à 28000.

Cette routine permet également l'affichage de mots-clés : ainsi, POKE 28001,255 provoquera l'affichage de **COPY.**

Afin de ne pas revenir au Basic pour déterminer la position PRINT, utilisons la routine de la MEM, située à 1601h, qui sélectionne le canal de sortie : on charge A avec 2 pour sélectionner l'écran, et avec 3 pour l'imprimante, puis on appelle 1601h, ce qui donne ainsi le choix pour afficher ou pour imprimer. De plus, une fois sélectionné, un canal de sortie, celui-ci est conservé pour le reste du programme C.M. ; en revanche, si l'on retourne au Basic, il faudra le sélectionner à nouveau. Par exemple, pour afficher A à la position (0,0), on peut entrer la succession des codes :

```
32000 3E,02        LD A,2        : canal 2
32002 CD,01,16     CALL 1601h    : sélectionne l'écran
32005 21,21,18     LD HL,1821h   : H=24 et L=33
32008 22,88,5C     LD (23688),HL: (S-POSN)
32011 3E,41        LD A,65       : code de A
32013 D7           RST 10
32014 C9           RET
```

On a chargé HL en une seule opération avec H=24 (numéro de ligne) et L=33 (numéro de colonne) ; ensuite, on entre L dans (23688) et H dans (23689), en une seule opération également. Ceci montre deux choses :

- la position PRINT (0,0) s'indique ici (24,33) : on

décompte les lignes à partir du haut et les colonnes à partir de la gauche ;

- la deuxième chose est que, en principe, on ne peut pas changer la variable système S-POSN, mais, il y a des exceptions.

RST 18

La variable système CH-ADD contient l'adresse du caractère suivant à interpréter. Cette routine extrait ce caractère et le place dans A.

RST 20

Cette routine a une action identique à RST 18, mais en passant un caractère (par exemple, un espace qui suit un mot-clé). Ces deux routines sont exploitées par l'interpréteur Basic.

RST 28

C'est la routine du calculateur en virgule flottante : un saut est fait à 13147 où se trouve la routine qui contrôle le calculateur. celle-ci extrait les octets qui suivent RST 28 pour effectuer les calculs demandés jusqu'à la rencontre de l'octet de valeur 56d qui en indique la fin.

RST 30

Cette routine crée l'espace souhaité dans la zone de travail temporaire qui se termine à STKBOT. Avant l'appel, on charge BC avec le nombre d'espaces souhaités. La routine utilise la variable système WORKSP, puis effectue un saut à 5790d où se trouve la routine qui augmente l'espace de travail temporaire

RST 38

C'est une routine d'interruption, que nous appelons INT, et qui sera démontée dans le paragraphe suivant

Les interruptions

Parmi les 13 bits qui constituent le bus des contrôles du microprocesseur Z80A (voir figure 1, chapitre I), il y en a trois : $\overline{\text{BUSRQ}}$, $\overline{\text{NMI}}$ et $\overline{\text{INT}}$ (voir connecteur arrière dans le manuel Sinclair) qui provoquent une interruption du microprocesseur. Les barres signifient que ces signaux sont actifs au niveau 0, en logique positive qui est adoptée dans le ZX6Spectrum.

Les trois interruptions possibles sont :

- la requête de bus : BUSQR. Cette interruption nécessite un dispositif DMA (accès direct à la mémoire) pour fonctionner et est prioritaire sur les deux autres ;
- l'interruption non masquable : MNI. Elle est toujours acceptée par le microprocesseur et possède la priorité 2. Au reçu d'un signal $\overline{\text{MNI}}$, le microprocesseur sauve le registre PC dans la pile, manipule les flip-flops internes pour empêcher toute interruption de 3ème ordre INT, puis force un saut à 66h -102d- ;
- l'interruption INT. Elle peut fonctionner selon trois modes différents. Comme, au cours de l'initialisation, le moniteur a mis le mode 1 (mnémonique IM1 de C.M. ED,56) ; seul, ce mode 1 sera considéré. Le microprocesseur sauve le registre PC dans la pile, puis effectue un saut à 38h -56d- tout comme RST 38, et manipule les flip-flops internes pour empêcher toute nouvelle INT.

La figure 1 du chapitre I montre le C.I. 1, logique Sinclair, qui joue un rôle essentiel pour l'appel de cette routine. Le Z80A fonctionne à une fréquence de 3,5 MHz, sa période T dure donc 0,2857 microseconde. Le C.I. 1 contient des compteurs : après 224 T, soit 64 microsecondes, le C.I. 1 envoie une impulsion de synchro-ligne vers la TV et après 70000 T, soit msec., il envoie des impulsions de synchro-trame vers la TV et, en même temps, un signal $\overline{\text{INT}}$ vers le microprocesseur. Celui-ci exécute alors le programme suivant :

```
56 F5        INT  PUSH AF
57 E5             PUSH HL
58 2A,78,5C       LD HL,(23672)   : (FRAMES)
61 23             INC HL          : incrémente FRAMES
62 22,78,5C       LD (23672),HL
65 7C             LD A,H
66 B5             OR L
67 20,03          JR NZ, +3+2
69 FD,34,40       INC (IY+64d)    : (FRAMES 3ème
                                    octet)
72 C5             PUSH BC
73 D5             PUSH DE
74 CD,BF,02       CALL 703d       : scrute le clavier
77 D1             POP DE
78 C1             POP BC
79 E1             POP HL
80 F1             POP AF
81 FB             EI
82 C9             RET
```

Le programme sauve les registres employés, puis les restaure avant le retour. Il incrémente les deux premiers octets de la variable système FRAMES et, si ces deux octets viennent à 0, il incrémente le 3ème octet de FRAMES. Il appelle enfin la routine de scrutation du clavier à 703d.

Une remarque intéressante est que la synchronisation verticale de l'image se fait en même temps que INT. En programmant un HALT, vous commencez la suite du programme directement en synchronisme avec l'image. Ce fait a été mis à profit dans l'exemple 2 du chapitre V.

ROUTINES DEMONTEES

La routine de scrutation du clavier

Cette routine, appelée par la routine INT, se développe de 703d à 797d. Elle fait appel à trois sous-routines :

- de 654 à 702d : SCRUT
- de 798 à 818d
- de 819 à 948d

La première des trois sous-routines est celle qui effectue la véritable scrutation et celle que nous avons appelée au cours des jeux du chapitre précédent. Au retour, DE contient un décodage des touches : la routine principale et les deux autres sous-routines transforment le retour de DE en code ASCII, exécutent la répétition d'une touche qui reste enfoncée, actualisent la variable système LAST-K et manipulent le bit 5 de FLAGS. LAST-K contient le code de la dernière touche enfoncée, tant qu'on ne pousse pas sur une nouvelle touche, d'où la nécessité d'un indicateur : le bit 5 de FLAGS est mis à 1 pour indiquer qu'une nouvelle touche a été pressée et remis à 0 quand aucune touche n'est pressée.

Démontons la première sous-routine :

```
654 2E,2F SCRUT  LD L,47d
656 11,FF,FF     LD DE,FFFF      : initialise DE
659 01,FE,FE     LD BC,FEFE      : adresse 1er port
662 ED,78  B1    IN A,(C)        : saisie
664 2F           CPL             : 1 bit à 1
665 E6,1F        AND 1F          : masque les trois
                                   bits hauts
667 28,0E        JR Z, +14+2     : saut si A=0
669 67           LD H,A
670 7D           LD A,L
```

```
671 14          ┌─►INC D
672 C0          │  RET NZ          : si trois touches
                │                    en même temps
673 D6,08   B2  │┌►SUB A,8
675 CB,3C       ││ SRL H           : Ci = H0
677 30,FA       │└─JR NC, -6+2     : saut si Ci = 0
679 53          │  LD D,E
680 5F          │  LD E,A
681 20,F4       └──JR NZ, -12+2    : saut s'il t a 2
                                     bits à 1 dans H
683 2D          ┌─►DEC L
684 CB,00       │  RLC B
686 38,E6       └──JR C, -26+2     : saut si Ci = 1
688 7A          ┌─►LD A,D
689 3C          │  INC A
690 C8          │  RET Z           : retour si D = 255
691 FE,28       │  CP 40d
693 C8          │  RET Z           : retour si D = 39
694 FE,19       │  CP 25d
696 C8          │  RET Z           : retour si D = 24
697 7B          │  LD A,E          : réalise EX D,E
698 5A          │  LD E,D
699 57          │  LD D,A
700 FE,18       │  CP 24d          : fait Z = 0
702 C9          └─►RET
```

Examinons la boucle B1 :

L'instruction IN A,(C) place le contenu du port C dans A par l'intermédiaire du C.I. 1, circuit spécialisé Sinclair, après que BC ait été placé sur le bus d'adresses. Comme BC = FEFE, ceci est l'adresse du 1er port du clavier et A8 du bus d'adresses est à 0.

L'instruction RLC B exécute une rotation à gauche de 8 bits : B7 va dans C_i et dans B0. A la 8ème rotation, B retrouve va valeur initiale et $C_i = 0$, ce qui provoque la sortie de la boucle. Celle-ci s'effectue donc 8 fois avec les valeurs de BC suivantes :

FEFE	met	A8 = 0
FDFE	met	A9 = 0
FBFE	met	A10 = 0
F7FE	met	A11 = 0
EFFE	met	A12 = 0
DFFE	met	A13 = 0
BFFE	met	A14 = 0
7FFE	met	A15 = 0

Le premier port met A8 = 0, les 5 touches CAPS SHIFT, Z, X, C, V sont à 0. Les 5 conducteurs B0 à B4, qui vont au CI 1, sont à 1. Quand on appuie sur une de ces 5

touches, alors qu'on scrute le premier port, on obtient :

```
- si on appuie sur CAPS SHIFT  B0=0  registre A : A0=0
- si on appuie sur Z           B1=0             : A1=0
- si on appuie sur X           B2=0             : A2=0
- si on appuie sur C           B3=0             : A3=0
- si on appuie sur V           B4=0             : A4=0
```

Arrivé à 665, le registre A a un bit à 1 si on a poussé sur une des 5 touches ci-dessus, sinon tous ses bits sont à 0. Dans ce dernier cas, on saute à DEC L, puis on saisit le 2ème port. Si on ne pousse sur aucune touche, à la sortie de la boucle B1, on a D=255 et E=255.

à 688, A=255
à 689, A=0
à 690, on retourne à celui qui a appelé cette routine avec pour *aucune touche pressée* : D=255, E=255 et Z=1.

Supposons qu'on pousse sur G (5ème touche du 2ème port). Après la saisie du 1er port, on a L=47-1=46 et, après la saisie du 2ème port, on a A=46-5x8=6. Au retour :

D=255, E=6 et Z=1

Pour résumer, si vous appuyez sur toutes les touches, sauf les deux SHIFT, il vient :

D=255, Z=1 et les valeurs de E suivantes :

```
E=0 pour B
  1 pour H
  2 pour Y
  3 pour 6
  4 pour 5
  5 pour T
  6 pour G
  7 pour V
E= de  8 à 15 pour N à C
E= de 16 à 23 pour M à X
E= de 24 à 31 pour SYmbol Shift à Z
E= de 32 à 39 pour SPACE à CAPS SHIFT
```

Pour visualiser les explications données, faîtes le programme :

```
28000 CD,8E,02   CALL 028E      : 205,142,2
28003 42         LD B,D         : 66
28004 4B         LD C,E         : 75
28005 C9         RET            : 201
```

Nous avons mis les valeurs décimales en observations : faîtes 6 POKE. Voyez avant (PRINT PEEK 28000) si vous n'avez rien à 28000 et seq., ce qui est le cas si votre ZX-Spectrum vient d'être mis sous tension.

Entrez ensuite le Basic :

```
10 PRINT AT 0,0 ; "Touche", "D  E  "
20 LET a$ = INKEY$
30 LET x=USR 28000
40 LET b=INT (x/256)
50 LET c=x-b*256
60 PRINT a$, b ; "  " ; c
70 PAUSE 100
90 GO TO 20
```

Poussez sur différentes touches.

La routine de production des sons

La routine SON, que nous démontons ci-après, est celle que nous avons employée au paragraphe SON du chapitre IV. Avant l'appel, on charge HL avec la hauteur du son désirée et DE avec la durée, celle-ci varie avec la fréquence.

La routine, appelée par la commande BEEP du Basic, fait appel à cette routine SON mais manipule, au préalable, le contenu de DE pour rendre la durée compatible avec la hauteur programmée.

```
949 F3      SON   DI              : INT inopérant
950 7D            LD A,L          : A=L
951 CB,3D         SRL L           : La=L/2 par défaut
953 CB,3D         SRL L           : Lb=La/2 par défaut
955 2F            CPL             : A=L
956 E6,03         AND 3           A=L modulo 4
958 4F            LD C,A          : C=A=0,1,2 ou 3
959 06,00         LD B,0
961 DD,21,D1,03   LD IX,977d      : on a besoin d'un
                                    registre de plus
965 DD,09         ADD IX,BC       : IX=977+0,1,2 ou 3
967 3A,48,5C      LD A,(23624)    : couleur des bords
                                    x 8
970 E6,38         AND 38h         : isole couleur bords
972 0F            RRCA            : met les bits 5,4,3
973 0F            RRCA              de A dans les bits
974 0F            RRCA              2,1,0 de A
975 F6,08         OR 8            : met A3=1
```

Arrêtons-nous un instant. Cette routine qui commence par DI et finit par EI empêche toute interruption INT pendant son exécution, donc le compteur FRAMES n'augmente pas et le clavier n'est pas scruté.

Le registre L est divisé deux fois par 2 par défaut. Par exemple, si L=255, il devient 127, puis 63. Cette

valeur finale de L sera employée plus loin pour charger C. Avant sa manipulation, on fait A=L, puis A=$\overline{L}$, puis A=$\overline{L}$ modulo 4, et cette dernière valeur est chargée finalement dans IX qui sera 977 ou 978 ou 979 ou 980.

Ensuite, de 967 à 974, on sauvegarde la couleur des bords pour qu'elle ne change pas, et à 975, on met A3 à 1 pour inhiber la sortie micro car on emploie, plus loin, le port de sortie FE.

```
 977 00      B1   NOP                                  4T
 978 00           NOP                                  4T
 979 00           NOP                                  4T
 980 04           INC B                                4T
 981 0C           INC C                                4T
 982 0D      B2   DEC C                                4T
 983 20,FD        JR NZ, -3+2                      12T/7T
 985 0E,3F        LD C,63d                             7T
 987 05           DEC B                                4T
 988 C2,D6,03     JP NZ, 982d                         10T
 991 EE,10        XOR 10h        : met A4 à 1
                                   puis à 0, etc.     7T
 993 D3,FE        OUT (FE),A     : vers H.P.         11T
 995 44           LD B,H         : charge B           4T
 996 4F           LD C,A         : sauve A            4T
 997 CB,67        BIT 4,A        : Z=$\overline{A4}$          8T
 999 20,09   B3   JR NZ, +9+2    : saut 1 fois
                                   sur 2          12T/7T
1001 7A           LD A,D                               4T
1002 B3           OR E                                 4T
1003 28,09   B4   JR NZ,+9+2     : saut si DE=0 12T/7T
1005 79           LD A,C         : restaure A         4T
1006 4D           LD C,L         : charge C           4T
1007 1B           DEC DE                               6T
1008 DD?E9        JP IX                                8T
1010 4D           LD C,L         : charge C           4T
1011 0C           INC C                                4T
1012 DD,E9        JP IX                                8T
1014 FB           EI             : restaure INT
1015 C9           RET
```

Les astuces de programmation se situent au niveau des durées. Voyons les deux temps possibles de la boucle partielle de 991 à 1012 :

	997 amène Z=0	997 amène Z=1
991	7T	7T
993	11T	11T
995	4T	4T
996	4T	4T

997	8T	8T	
999	12T	7T	
1001		4T	
1002		4T	
1003		7T	
1005		4T	
1006		4T	
1007		6T	actualise DE
1008		8T	
1010	4T		
1011	4T		
1012	8T		
	62T	78T	
+ une boucle C	16T		

1007 actualise DE une fois sur deux, donc à chaque période de son. En 1011, il y a INC C donc, dans cette éventualité, la boucle C (982 et 983) est effectuée une fois de plus que dans l'autre éventualité, donc les durées sont toutes deux égales à 78T.

Les commandes Basic

Quand, à la mise sous tension, l'ordinateur a terminé sa propre initialisation, il entre dans une boucle d'exécution qui se répète continuellement. Cette boucle, comme toutes les routines auxquelles elle fait appel, est interrompue 50 fois par seconde, sauf pour les routines et programmes qui ont programmé DI et EI. Cette interruption permet la saisie d'une entrée au clavier. Quelle que soit cette entrée, il est fait appel à l'interpréteur Basic pour exécuter l'une des 50 commandes Basic.

A un certain endroit, il y a les lignes suivantes pour orienter vers la routine adéquate : avant ces lignes, A contient le code caractère (commande Basic) et B=0.

```
6980 D6,CE        SUB A,206d     : A=A-206
6982 DA,8A,1C     JP C,7306      : pour afficher un
                                   code d'erreur
6985 4F           LD C,A
6986 21,48,1A     LD HL,6728     : table des déplacements
6989 09           ADD HL,BC      : HL pointe sur le
                                   déplacement
6990 4E           LD C,(HL)      : saisie du déplacement
6991 09           ADD HL,BC      : HL pointe sur la
                                   table de syntaxe.
```

A 6728, il y a une table de déplacements qui s'étend jusqu'à 6777, puisqu'il y a 50 commandes possibles. Cette table contient le déplacement à ajouter pour pointer dans la table de syntaxe qui va de 6778 à 6934. Cette table donne une ou plusieurs classes de syntaxe parmi 12 auxquelles un ou plusieurs arguments doivent se soumettre pour la vérification syntaxique ; elle donne la ponctuation exigée et, enfin, l'adresse de la routine.

Ceci est résumé dans le tableau suivant :

Commande	*Code*	*Adresse déplac.*	*(Adresse déplac.)*	*Adresse syntaxe*	*Adresse routine*
DEF FN	206	6728	177	6905	8032
CAT	207	6729	203	6932	6035
FORMAT	208	6730	188	6918	6035
MOVE	209	6731	191	6922	6035
ERASE	210	6732	196	6928	6035
OPEN *#*	211	6733	175	6908	5942
CLOSE *#*	212	6734	180	6914	5861
MERGE	213	6735	147	6882	1541
VERIFY	214	6736	145	6881	1541
BEEP	215	6737	146	6883	1016
CIRCLE	216	6738	149	6887	8992
INK	217	6739	152	6891	7318
PAPER	218	6740	152	6892	7318
FLASH	219	6741	152	6893	7318
BRIGHT	220	6742	152	6894	7318
INVERSE	221	6743	152	6895	7318
OVER	222	6744	152	6896	7318
OUT	223	6745	152	6897	7802
LPRINT	224	6746	127	6873	8137
LLIST	225	6747	129	6876	6133
STOP	226	6748	46	6794	7406
READ	227	6749	108	6857	7661
DATA	228	6750	110	6860	7719
RESTORE	229	6751	112	6863	7746
NEW	230	6752	72	6824	4535
BORDER	231	6753	148	6901	8852
CONTINUE	232	6754	86	6840	7775
DIM	233	6755	63	6818	11266
REM	234	6756	65	6821	7090
FOR	235	6757	43	6800	7427
GO TO	236	6758	23	6781	7783
GO SUB	237	6759	31	6790	7917
INPUT	238	6760	55	6815	8329
LOAD	239	6761	119	6880	1541
LIST	240	6762	68	6830	6137

Commande	*Code*	*Adresse déplac.*	*(Adresse déplac.)*	*Adresse syntaxe*	*Adresse routine*
LET	241	6763	15	6778	11007
PAUSE	242	6764	89	6853	7994
NEXT	243	6765	43	6808	7595
POKE	244	6766	67	6833	7808
PRINT	245	6767	45	6812	8141
PLOT	246	6768	81	6849	8924
RUN	247	6769	58	6827	7841
SAVE	248	6770	109	6879	1541
RANDOMIZE	249	6771	66	6837	7759
IF	250	6772	13	6785	7408
CLS	251	6773	73	6846	3435
DRAW	252	6774	92	6866	9090
CLEAR	253	6775	68	6843	7852
RETURN	254	6776	21	6797	7971
COPY	255	6777	93	6870	3756

Les calculs en virgule flottante

Le RST 28 de C.M. EF oblige un saut à 13147 où se trouve la routine qui contrôle les calculs en virgule flottante. Cette routine s'étend jusqu'à 13217. Elle extrait les octets qui suivent RST 28 dont la valeur correspond à une opération, un calcul. QUand ce calcul porte sur un opérande, on fait pointer HL sur (STKEND) -5 et, pour un calcul sur deux opérandes, sur (STKEND) -10. Le code du calcul est multiplié par 2, puis additionné à 13015 où se trouve le début d'une table qui s'étend jusqu'à 13146. Cette table donne l'adresse de la routine correspondant au code de l'opération. L'extraction des octets continue jusqu'à la rencontre de 38h -56d- qui est le DEF de FIN des calculs. Ce code donne l'adresse 369B-13979d-.

```
13979 F1    FIN   POP AF          : actualise SP
13980 D9          EXX             : 2ème jeu des
                                    registres
13981 E3          EX (SP),HL      : place H'L' dans SP
13982 D9          EXX
13983 C9          RET
```

POP AF a pour fonction d'actualiser la pile car celle-ci tient, à ce moment, une adresse de retour dans la routine de contrôle. On fait EXX car la routine de contrôle emploie le 2ème jeu de registres, et c'est H'L' qui tient l'adresse de l'octet qui suit DEF FIN ; cette adresse est poussée dans la pile et RET fait retour à cette adresse.

Nous donnons, ci-après, le contenu de la table des adresses, ainsi que les codes à employer pour réaliser une opération.

Tables d	*Adresse h*	*Routine*	*Code opération h*
13015	368F		
13017	343C		
13019	33A1		
13021	300F	-	03
13023	30CA	*	04
13025	31AF	/	05
13027	3851	↑	06
13029	351B	OU	07
13031	3524		
13033	353B		
"	"		
13045	3014	+	0F
13047	352D		
13049	353B		
"	"		
13061	359C		
13063	35DE		
13065	34BC		
13067	3645		
13069	346E		
13071	3669	CODE	1C
13073	35DE	VAL	1D
13075	3674	LEN	1E
13077	37B5	SIN	1F
13079	37AA	COS	20
13081	37DA	TAN	21
13083	3833	ASN	22
13085	3843	ACS	23
13087	37E2	ATN	24
13089	3713	LN	25
13091	36C4	EXP	26
13093	36AF	INT	27
13095	384A	SQR	28
13097	3492	SGN	29
13099	346A	ABS	2A
13101	34AC	PEEK	2B
13103	34A5	IN	2C
13105	34B3	USR	2D
13107	361F	STR$	2E
13109	35C9	CHR$	2F
13111	3501	NOT	30
13113	33C0		
13115	36A0		

Tables *d*	*Adresse* *h*	*Routine*	*Code opération* *h*
13117	3686		
13119	33C6		
13121	367A		
13123	3506		
13125	34F9		
13127	369B	DEF FIN	38

La routine de contrôle exploite les nombres, en virgule flottante, présents dans la pile calculateur. Pour les y placer, on fait appel à une routine située à 11560.

Cette routine a deux points d'entrée : nous l'appelons PILEA ou PILEBC, suivant que l'on entre dans la pile calculateur un nombre plus petit que 256 ou un nombre plus petit que 65536 ; exemple :

```
LD BC,x
CALL PILEBC
```

Quand le calculateur a terminé les calculs demandés, le résultat se trouve dans la pile calculateur. Nous pouvons le transférer dans le registre BC, à l'aide d'une routine que nous appelons DEPILE, située à 11682.

Par exemple, faisons l'opération 64 x 200.

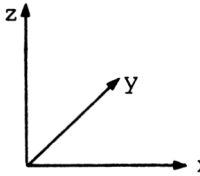
```
32000 01,40,00   LD BC,64
32003 CD,2B,2D   CALL PILEBC
32006 3E,C8      LD A,200
32008 CD,28,2D   CALL PILEA
32011 EF         RST 28
32012 04         DEF MULT
32013 38         DEF FIN
32014 CD,A2,2D   CALL DEPILE
32017 C9         RET
```

PRINT USR 32000 vous donne 12800.

Il est évident que DEPILE, qui retourne une valeur dans BC, ne peut remplir BC qu'avec une valeur plus petite que 65536. Pour avoir la valeur exacte qui se trouve dans la pile calculateur, on la transfère dans une variable qui aura été déclarée dans un programme Basic et exécutée.

Graphique de courbe en trois dimensions

A titre d'illustration, dessinons un graphique en relief, à l'aide des coordonnées en trois dimensions.



L'origine et les directions positives ont été pris pour respecter le système du ZX-Spectrum, avec x < 256 et z < 176. Définissons une fonction z=f(x,y) qui donne des valeurs positives : une des deux variables indépendantes peut être nulle.

Si vous voulez du relief, faîtes :

```
PLOT x+y,z
```

si vous voulez de la perspective, faîtes :

```
PLOT x+y,z+y
```

évidemment, avec x+y<256 et z+y<176.

Commençons à exécuter le graphique Basic :

```
10 FOR y=0 TO 63 STEP 7
20 FOR x=4 TO 194 STEP 2
30 LET z=SIN (x*314/6500)* SIN (y*314/7000)*85
40 PLOT x+y-4,z+87
50 NEXT x
60 NEXT y
100 STOP
```

Faîtes RUN et appréciez la lenteur du déroulement.

Le programme qui suit, entièrement en code machine, permet de ganger un peu de temps :

```
32000 00          DEF x
32001 00          DEF y
32002 3E,57 CALC  LD A,87d      : 5ème article
32004 CD,28,2D    CALL PILEA
32007 3E,55       LD A,85d      : 4ème article
32009 CD,28,2D    CALL PILEA
32012 3A,00,7D    LD A,(32000)  : x = 3ème article
32015 CD,28,2D    CALL PILEA
32018 01,3A,01    LD BC,314     : 2ème article
32021 CD,2B,2D    CALL PILE BC
32024 01,64,19    LD BC,6500    : 1er article
32027 CD,2B,2D    CALL PILEBC
32030 EF          RST 28
32031 05          DEF DIV
32032 04          DEF MULT
32033 1F          DEF SIN
32034 04          DEF MULT
```

```
32035 38          DEF FIN
32036 3A,01,7D    LD A,(32001)   : y = 3ème article
32039 CD,28,2D    CALL PILE A
32042 01,3A,01    LD BC,314      : 2ème article
32045 CD,2B,2D    CALL PILEBC
32048 01,58,1B    LD BC,7000     : 1er article
32051 CD,2B,2D    CALL PILEBC
32054 EF          RST 28
32055 05          DEF DIV
32056 04          DEF MULT
32057 1F          DEF SIN
32058 04          DEF MULT
32059 0F          DEF ADD
32060 38          DEF FIN
32061 CD,A2,2D    CALL DEPILE
32064 C9          RET
32065 1E,00
        GRAPHE    LD E,0         : y
32067 16,04       LD D,4         : x
32069 7B          LD A,E
32070 32?01,7D    LD(32001),A    : place y dans 32001
32073 7A          LD A,D
32074 32,00,7D    LD(32000),A    : place x dans 32000
32077 D5          PUSH DE        : sauve x,y
32078 CD,02,7D    CALL CALC      : 32002
32081 D1          POP DE         : restaure x,y
32082 41          LD B,C         : charge z dans B
32083 3E,FC       : LD A,252d    : soit -4
32085 83          ADD A,E        : A=y-4
32086 82          ADD A,D        : A=x+y-4
32087 4F          LD C,A         : charge x dans C
32088 D5          PUSH DE        : sauve x,y
32089 CD,DF,22    CALL PLOT      : 8927d
32092 D1          POP DE         : restaure x,y
32093 14          INC D
32094 14          INC D          : x=x+2
32095 7A          LD A,D
32096 FE,C4       CP 196d
32098 20,E1       JR NZ, -31+2   : saut si x<196
32100 3E,07       LD A,7
32102 83          ADD A,E
32103 5F          LD E,A         : y=y+7
32104 FE,46       CP 70
32106 20,D7       JR NZ, -41+2   : saut si y<70
32108 C9          RET
```

(RANDOMIZE USR 32065)

Pour aller vraiment plus vite, il faut faire une table de sinus que l'on place dans des octets réservés et on remplace la sous-routine CALC par une routine qui ne fait pas d'appel à la MEM.

ANNEXE I - table 1

Liste alphabétique des mnémoniques

Leurs codes, les instructions qu'ils représentent. Le nombre d'octets nécessaires se trouve dans la colonne 2 (codes).

Mnémoniques	*Codes*	*Instructions*
ADC A,R	Table 2	A=A+R+C_i (C_i=indicateur C)
ADC A,(RR)	Table 2	A=A+(RR)+C_i
ADC A,n	CE,nn	A+A+n+C_i
ADC HL,RR	Table 3	HL=HL+RR+C_i
ADD A,R	Table 2	A=A+R
ADD A,(RR)	Table 2	A=A+(RR)
ADD A,n	C6,nn	A=A+n
ADD HL,RR	Table 3	HL=HL+RR
ADD IX,RR	Table 3	IX=IX+RR
ADD IY,RR	Table 3	IY=IY+RR
AND R	Table 2	A=A R
AND (RR)	Table 2	A=A (RR)
AND n	E6,nn	A=A n
BIT b,R	Table 2	Z=$\overline{R}_b$
BIT b,(RR)	Table 2	Z=$(\overline{RR})_b$
CALL NZ,pq	C4,qq,pp	si Z=0)
CALL Z,pq	CC,qq,pp	si Z=1) alors
CALL NC,pq	DA,qq,pp	si C=0) (SP-1)=PC haut
CALL C,pq	DC,qq,pp	si C+1) (SP-2)=PC haut

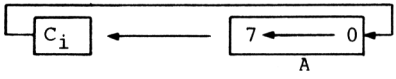
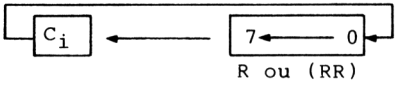
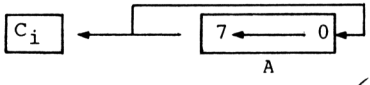
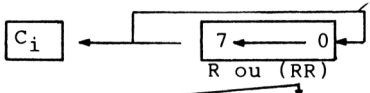
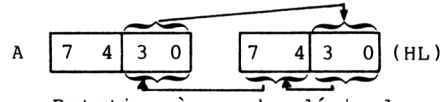
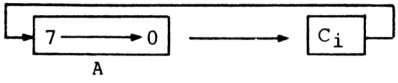
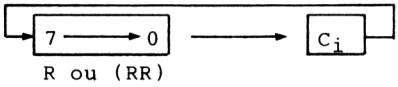
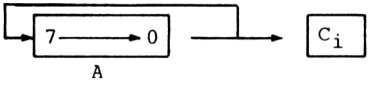
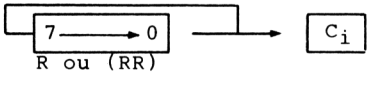
Mnémoniques	*Codes*	*Instructions*
CALL PO,PQ	E4,qq,pp	si P/V=0) SP=SP-2
CALL PE,PQ	EC,qq,pp	P/V=1) PC=pq
CALL P,pq	F4,qq,pp	si S=0)
CALL P,pq	FC,qq,pp	si S=1)
		si non alors PC=PC+3
CALL pq	CD,qq,pp	(SP-1)=PC haut (SP-2)=PC bas SP=SP-2 et PC=pq
CCF	3F	C =$\overline{C}_i$ H devient le C_i précédent
CP R	Table 2	A-R
CP (RR)	Table 2	A-(RR)
CP n	FE,nn	A-n
CPD	ED,A9	A-(HL) HL=HL-1 BC=BC-1
CPDR	ED,B9	A-(HL) P/V=0 si BC=0 HL=HL-1 après exécution BC=BC-1 si non P/V=1 Répétez jusqu'à ce que BC=0 ou A=(HL)
CPI	ED,A1	A-(HL) Z=1 si HL=HL+1 A=(HL) BC=BC-1
CPIR	ED,B*	A-(HL) HL=HL+1 BC=BC-1 Répétez jusqu'à ce que BC=0 ou A=(HL)
CPL	2F	A=$\overline{A}$ (complément à 1)
DAA	27	Effectue différents ajustements arithmétiques selon les indicateurs N,H et C (employé en dcb)
DEC R	Table 2	R=R-1
DEC (RR)	Table 2	(RR)=(RR)-1
DEC RR	Table 3	RR=RR-1
DI	F3	IFF=0 mise hors d'action des interruptions masquables

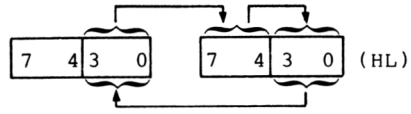
Mnémoniques	*Codes*	*Instructions*
DJNZ e	10,ee	B=B-1 si B≠0, faire PC=PC+e+2 e est un nombre avec signe
EI	FB	IFF=1, remise en action des interruptions masquables
EX AF,AF'	08	A = A' et F = F' échange
EX DE,HL	EB	DE = HL échange
EX (SP),RR	Table 3	(SP) = RRbs (SP+1) = RRhaut } échange
EXX	D9	BC = BC' DE = DE' HL = HL' } échange
HALT	76	Suspend le microprocesseur. Celui-ci effectue des NOP jusqu'à réception d'une interruption ou d'un reset.
IM 0	ED?46	Met le mode 0 d'interruption, le dispositif envoie une instruction.
IM 1	ED,56	Met le mode 1 d'interruption, un RST 38h sera exécuté.
IM 2	ED,5E	Met le mode 2 d'interruption, le dispositif envoie un octet considéré comme octet bas d'une adresse. Son octet haut est pris dans I.
IN R, (C)	Table 2	R=(C) Le contenu de C fournit l'octet bas de l'adresse du port. B fournit l'octet haut.
IN A,(p)	DB,pp	A=p) p est l'octet bas de l'adresse du port. A fournit l'octet haut.
INC R	Table 2	R=R+1
INC (RR)	Table 2	(RR)=(RR)+1
INC RR	Table 3	RR=RR+1
IND	ED,AA	(HL)=(C) Z=1 si B=0 B=B-1 après exécution, HL=HL-1 sinon Z=0 Entrée du port adressé par C avec DEC.

Mnémoniques	*Codes*	*Instructions*
INDR	ED,BA	(HL)=(C) B=B-1 HL=HL-1 Répétez jusqu'à ce que B=0. Entrée de bloc du port adressé par C avec DEC.
INI	ED,A2	(HL)=(C) Z=1 si B=0 B=B-1 après exécution HL=HL+1 sinon Z=0 Entrée du port adressé par C avec INC.
INIR	ED,B2	(HL)=(C) B=B-1 HL=HL-1 Répétez jusqu'à ce que B=0. Entrée de bloc du port adressé par C avec INC.
JP NZ,pq	C2,qq,pp	si Z=0
JP Z,pq	CA,qq,pp	si Z=1
JP NC,pq	D2,qq,pp	si C=0
JP C,pq	DA,qq,pp	si C=1
JP PO,pq	E2,qq,pp	si P/V=0
JP PE,pq	EA,qq,pp	si P/V=1
JP P,pq	F2,qq,pp	si S=0
JP M,pq	FA,qq,pp	si S=1 alors PC=pq si non, alors PC=PC+3
JP pq	C3,qq,pp	PC=pq
JP RR	Table 3	PC=RR
JR NZ,e	20,ee	si Z=0
JR Z,e	28,ee	si Z=1
JR NC,e	30,ee	si C=0
JR C,e	38,ee	si C=1 alors PC=PC+e+2 si non PC=PC+2 e est un nombre avec signe.
JR e	18,ee	PC=PC+e+2
LD RR,(nm)	Table 3	RRbas=(nm) RRhaut=(nm+1)
LD (nm),RR	Table 3	(nm)=RRbas (nm+1)=RRhaut
LD RR,nm	Table 3	RR=nm
LD SP,RR	Table 3	SP=RR
LD R,n	Table 2	R=n
LD (RR),n	Table 2	(RR)=n

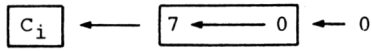
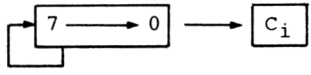
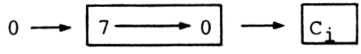
Mnémoniques	*Codes*	*Instructions*
LD A,R ou RR	Table 2	A=R ou A=(RR)
LD B,R ou (RR)	Table 2	B=R ou B=(RR)
LD C,R ou (RR)	Table. 2	D=R ou D=(RR)
LD E,R ou (RR)	Table 2	E=R ou E=(RR)
LD H,R ou (RR)	Table 2	H=R ou H=(RR)
LD L,R ou (RR)	Table 2	L=R ou L=(RR)
LD (HL),R	Table 2	(HL)=R
LD (IX+d),R	Table 2	(IX+d)=R
LD (IY+d),R	Table 2	(IY+d)=2
LD (BC),A	02	(BC)=A
LD (DE),A	12	(DE)=A
LD (nm),A	32,mm,nn	(nm)=A
LD A,(BC)	0A	A=(BC)
LD A,(DE)	1A	A=(DE)
LD A,(nm)	3A,mn,nn	A=(nm)
LD I,A	ED,47	I=A
LD R,A	ED,4F	R=A
LD A,I	ED,57	A=I P/V=IFF2
LD A,R	ED,5F	A=R
LDD	ED,A8	(DE)=(HL) DE=DE-1 HL=HL-1 BC=BC-1 P/V=0 si BC=0 après exécution
LDI	ED,A0	(DE)=(HL) sinon P/V=1 DE=DE+1 HL=HL+1 BC=BC-1 Chargement d'octet avec DEC ou INC
LDDR	ED,B8	(DE)=(HL) DE=DE-1 HL=HL-1 BC=BC-1 Répétez jusqu'à ce que BC=0.
LDIR	ED,B0	(DE)=(HL) DE=DE+1 HL=HL+1 BC=BC-1 Répétez jusqu'à ce que BC=0 chargement de bloc avec DEC ou INC.
NEG	ED,44	A=0-A (complément à 2)

Mnémoniques	*Codes*	*Instructions*
NOP	00	Pas d'opération.
OR R	Table 2	A=A V R
OR (RR)	Table 2	A=A V (RR)
OR n	F6,nn	A=A V n
OTDR	EB,BB	(C)=(HL) B=B-1 HL=HL-1 Répétez jusqu'à ce que B=0
OTIR	ED,B3	(C)=(HL) B=B-1 HL=HL+1 Répétez jusqu'à ce que B=0. Sortie de bloc avec DEC ou INC C fournit l'octet bas de l'adresse du prot B après décrémentation fournit l'octet haut.
OUTD	ED,AB	(C)=(HL) B=B-1 HL=HL-1 Z=1 si B=0 après exécution sinon Z=0
OUTI	ED,A3	(C)=(HL) B=B-1 HL=HL+1 Sortie d'octet avec DEC ou INC.
OUT (C),R	Table 2	(C)=2 R est mis dans le port adresse par C pour l'octet bas et par B pour le haut.
OUT (p),A	D3,pp	(p)=A A est mis dans le port adressé par p.
POP RR	Table 3	RRbas =(SP) RRhaut =(SP+1) SP=SP+2
PUSH RR	Table 3	(SP-1)=RRhaut (SP-2)=RRbas SP=SP-2
RES b,R	Table 2	R_b =0
RES b,(RR)	Table 2	$(RR)_b$ =0
RET	C9	PCbas=(SP) PChaut=(SP+1) SP=SP+2
RETI	ED,4D	PCbas=(SP) retour d'interruption PChaut=(SP+1) EI doit être avant RETI SP=SP+2

Mnémoniques	*Codes*	*Instructions*
RETN	ED,45	PCbas=(SP) retour d'interruption PChaut=(SP+1) non masquable SP=SP+2 IFF1=IFF2
RET NZ	C0	si Z=0
RET Z	C8	si Z=1
RET NC	D0	si C=0
RET C	D8	si C=1
RET PO	E0	si P/V=0
RET PE	E8	si P/V=1
RET P	F0	si S=0
RET M	F8	si S=1
		alors PCbas=(SP) PChaut=(SP+1) SP=SP+2
		sinon alors PC=PC+1
RLA	17	
RL R ou (RR)	Table 2	
RLCA	07	
RLC R ou (RR)	Table 2	
RLD	ED,6F	
RRA	1F	
RR R ou (RR)	Table 2	
RRCA	OF	
RRC R ou (RR)		

Mnémoniques	*Codes*	*Instructions*
RRD	ED,67	 Rotation à droite décimale (par quartet - pour DCB)
RST 00	C7	p=00
RST 08	CF	p=08
RST 10	D7	p=10f
RST 18	DF	p=18h
RST 20	E7	p=20h
RST 28	EF	p=28h
RST 30	F7	p=30h
RST 38	FF	p=38h

(SP-1)=PChaut
(SP-2)=PCbas
SP=SP-2
PChaut=0
PCbas =p

SUB A,R	Table 2	A=A-R
SUB A,(RR)	Table 2	A=A-(RR)
SUB A,n	D6,nn	A=A-n
SBC A,R	Table 2	A=A-R-C_i
SBC A,(RR)	Table 2	A=A-(RR)-C_i
SBC A,n	DE,nn	A=A-n-C_i
SBC HL,RR	Table 3	HL=HL-RR-C_i
SCF	37	C_i=1
SET b,R	Table 2	R_b=1
SET b,(RR)	Table 2	(RR)$_b$=1
SLA R ou (RR)	Table 2	 R ou (RR) Glissement à gauche arithmétique
SRA R ou (RR)	Table 2	 Glissement à droite arithmétique
SRL R ou (RR)	Table 2	 Glissement à droite logique
XOR R	Table 2	A=A⊻ R
XOR (RR)	Table 2	A=A⊻ (RR)
XOR n	EE,nn	A=A⊻ n

ANNEXE I

Liste alphabétique des mnémoniques avec leurs codes

Mnémoniques	*R*			
	A	*B*	*C*	*D*
ADC A,R ADC A,(RR)	8F	88	89	8A
ADD A,R ADD A,(RR)	87	80	81	82
AND R AND (RR)	A7	A0	A1	A2
BIT 0,R ou (RR)	CB,47	CB,40	CB,41	CB,42
BIT 1,R ou (RR)	CB,4F	CB,48	CB,49	CB,4A
BIT 2,R ou (RR)	CB,57	CB,50	CB,51	CB,52
BIT 3,R ou (RR)	CB,5F	CB,58	CB,59	CB,5A
BIT 4,R ou (RR)	CB,67	CB,60	CB,61	CB,62
BIT 5,R ou (RR)	CB,6F	CB,68	CB,69	CB,6A
BIT 6,R ou (RR)	CB,77	CB,70	CB,71	CB,72
BIT 7,R ou (RR)	CB,7F	CB,78	CB,79	CB,7A
CP R CP (RR)	BF	B8	B9	BA
DEC R DEC (RR)	3D	05	0D	15
IN R,(C)	ED,78	ED,40	ED,48	ED,50
INC R INC (RR)	3C	04	0C	14
LD R,n LD (RR),n	3E,nn	06,nn	0E,nn	16,nn
LD A,R ou (RR)	7F	78	79	7A
LD B,R ou (RR)	47	40	41	42
LD C,R ou (RR)	4F	48	49	4A
LD D,R ou (RR)	57	50	51	52

- table 2

(qui montrent le nombre d'octets nécessaires).

			(RR)		
E	*H*	*L*	*(HL)*	*(IX+d)*	*(IY+d)*
8B	8C	8D	8E	DD,8E,dd	FD,8E,dd
83	84	85	86	DD,86,dd	FD,86,dd
A3	A4	A5	A6	DD,A6,dd	FD,A6,dd
CB,43	CB,44	CB,45	CB,46	DD,CB,dd,46	FD,CB,dd,46
CB,4B	CB,4C	CB,4D	CB,4E	DD,CB,dd,4E	FD,CB,dd,4E
CB,53	CB,54	CB,55	CB,56	DD,CB,dd,56	FD,CB,dd,56
CB,5B	CB,5C	CB,5D	CB,5E	DD,CB,dd,5E	FD,CB,dd,5E
CB,63	CB,64	CB,65	CB,66	DD,CB,dd,66	FD,CB,dd,66
CB,6B	CB,6C	CB,6D	CB,6E	DD,CB,dd,6E	FD,CB,dd,6E
CB,73	CB,74	CB,75	CB,76	DD,LB,dd,76	FD,CD,dd,76
CB,7B	CB,7C	CB,7D	CB,7E	DD,CB,dd,7E	FD,CB,dd,7E
BB	BC	BD	BE	DD,BE,dd	FD,BE,dd
1D	25	2D	35	DD,35,dd	FD,35,dd
ED,58	ED,60	ED,68			
1C	24	2C	34	DD,34,dd	FD,34,dd
1E,nn	26,nn	2E,nn	36,nn	DD,36,dd,nn	FD,36,dd,nn
7B	7C	7D	7E	DD,7E,dd	FD,7E,dd
43	44	45	46	DD,46,dd	FD,46,dd
4B	4C	4D	4E	DD,4E,dd	FD,4E,dd
53	54	55	56	DD,56,dd	FD,56,dd

ANNEXE I - table 2 (suite)

Mnémoniques	*R*			
	A	*B*	*C*	*D*
LD E,R ou (RR)	5F	58	59	5A
LD H,R ou (RR)	67	60	61	62
LD L,R ou (RR)	6F	68	69	6A
LD (HL),R	77	70	71	72
LD (IX+d),R	DD,77,dd	DD,70,dd	DD,71,dd	DD,72,dd
LD (IY+d),R	FD,77,dd	FD,70,dd	FD,71,dd	FD,72,dd
OR R OR (RR)	B7	B0	B1	B2
OUT (C),R	ED,79	ED,41	ED,49	ED,51
RES 0,R ou (RR)	CB,87	CB,80	CB,81	CB,82
RES 1,R ou (RR)	CB,8F	CB,88	CB,89	CB,8A
RES 2,R ou (RR)	CB,97	CB,90	CB,91	CB,92
RES 3,R ou (RR)	CB,9F	CB,98	CB,99	CB,9A
RES 4,R ou (RR)	CB,A7	CB,A0	CB,A1	CB,A2
RES 5,R ou (RR)	CB,AF	CB,A8	CB,A9	CB,AA
RES 6,R ou (RR)	CB,B7	CB,B0	CB,B1	CB,B2
RES 7,R ou (RR)	CB,BF	CB,B8	CB,B9	CB,BA
RL R ou (RR)	CB,17	CB,10	CB,11	CB,12
RLC R ou (RR)	CB,07	CB,00	CB,01	CB,02
RR R ou (RR)	CB,1F	CB,18	CB,19	CB,1A
RRC R ou (RR)	CB,0F	CB,08	CB,09	CB,0A
SUB A,R SUB A,(RR)	97	90	91	92
SBC A,R SBC A,(RR)	9F	98	99	9A
SET 0,R ou (RR)	CB,C7	CB,C0	CB,C1	CB,C2
SET 1,R ou (RR)	CB,CF	CB,C8	CB,C9	CB,CA
SET 2,R ou (RR)	CB,D7	CB,D0	CB,D1	CB,D2
SET 3,R ou (RR)	CB,DF	CB,D8	CB,D9	CB,DA
SET 4,R ou (RR)	CB,E7	CB,E0	CB,E1	CB,E2
SET 5,R ou (RR)	CB,EF	CB,E8	CB,E9	CB,EA
SET 6,R ou (RR)	CB,F7	CB,F0	CB,F1	CB,F2
SET 7,R ou (RR)	CB,FF	CB,F8	CB,F9	CB,FA
SLA R ou (RR)	CB,27	CB,20	CB,21	CB,22
SRA R ou (RR)	CB,2F	CB,28	CB,29	CB,2A
SRL R ou (RR)	CB,3F	CB,38	CB,39	CB,3A
XOR R XOR (RR)	AF	A8	A9	AA

			(RR)		
E	*H*	*L*	*(HL)*	*(IX+d)*	*(IY+d)*
5B	5C	5D	5E	DD,5E,dd	FD,5E,dd
63	64	65	66	DD,66,dd	FD,66,dd
6B	6C	6D	6E	DD,6E,dd	FD,6E,dd
73	74	75			
DD,73,dd	DD,74,dd	DD,75,dd			
FD,73,dd	DF,74,dd	DF,75,dd			
B3	B4	B5			
			B6	DD,B6,dd	FD,B6,dd
ED,59	ED,61	ED,69			
CB,83	CB,84	CB,85	CB,86	DD,CB,dd,86	FD,CB,dd,86
CB,8B	CB,8C	CB,8D	CB,8E	DD,CB,dd,8E	FD,CB,dd,8E
CB,93	CB,94	CB,95	CB,96	DD,CB,dd,96	FD,CB,dd,96
CB,9B	CB,9C	CB,9D	CB,9E	DD,CB,dd,9E	FD,CB,dd,9E
CB,A3	CB,4A	CB,A5	CB,A6	DD,CB,dd,A6	FD,CB,dd,A6
CB,AB	CB,AC	CB,AD	CB,AE	DD,CB,dd,AE	FD,CB,dd,AE
CB,B3	CB,B4	CB,B5	CB,B6	DD,CB,dd,B6	FD,CB,dd,B6
CB,BB	CB,BC	CB,BD	CB,BE	DD,CB,dd,BE	FD,CB,dd,BE
CB,13	CB,14	CB,15	CB,16	DD,CB,dd,16	FD,CB,dd,16
CB,03	CB,04	CB,05	CB,06	DD,CB,dd,06	FD,CB,dd,06
CB,1B	CB,1C	CB,1D	CB,1E	DD,CB,dd,1E	FD,CB,dd,1E
CB,0B	CB,0C	CB,0D	CB,0E	DD,CB,dd,0E	FD,CB,dd,0E
93	94	95			
			96	DD,96,dd	FD,96,dd
9B	9C	9D			
			9E	DD,9E,dd	FD,9E,dd
CB,C3	CB,C4	CB,C5	CB,C6	DD,CB,dd,C6	FD,CB,dd,C6
CB,CB	CB,CC	CB,CD	CB,CE	DD,CB,dd,CE	FD,CB,dd,CE
CB,D3	CB,D4	CB,D5	CB,D6	DD,CB,dd,D6	FD,CB,dd,D6
CB,DB	CB,DC	CB,DD	CB,DE	DD,CB,dd,DE	FD,CB,dd,DE
CB,E3	CB,E4	CB,E5	CB,E6	DD,CB,dd,E6	FD,CB,dd,E6
CB,EB	CB,EC	CB,ED	CB,EE	DD,CB,DD,EE	FD,CB,dd,EE
CB,F3	CB,F4	CB,F5	CB,F6	DD,CB,dd,F6	FD,CB,dd,F6
CB,FB	CB,FC	CB,FD	CB,FE	DD,CB,dd,FE	FD,CB,dd,FE
CB,23	CB,24	CB,25	CB,26	DD,CB,dd,26	FD,CB,dd,26
CB,2B	CB,2C	CB,2D	CB,2E	DD,CB,dd,2E	FD,CB,dd,2E
CB,3B	CB,3C	CB,3D	CB,3E	DD,CB,dd,3E	FD,CB,dd,3E
AB	AC	AD			
			AE	DD,AE,dd	FD,AE,dd

ANNEXE I- table 3

Liste alphabétique des mnémoniques avec leurs codes (qui montrent le nombre d'octets nécessaires)

Mnémoniques	*RR*						
	AF	*BC*	*DE*	*HL*	*IX*	*IY*	*SP*
ADC HL,RR		ED,4A	ED,5A	ED,6A			ED,7A
ADD HL,RR		09	19	29			39
ADD IX,RR		DD,09	DD,19		DD,29		DD,39
ADD IY,RR		FD,09	FD,19			FD,29	FD,39
DEC RR		0B	1B	2B	DD,2B	FD,2B	3B
EX (SP),RR				E3	DD,E3	FD,E3	
INC RR		03	13	23	DD,23	FD,23	33
JP RR				E9	DD,E9	FD,E9	
LR RR,(nm)		ED,4B,mm,nn	ED,5B,mm,nn	ED,6B,mm,nn			ED,7B,mm,nn
LD RR,(nm)				2A,mm,nn	DD,2A,mm,nn	FD,2A,mm,nn	
LD (nm),RR		ED,43,mm,nn	ED,53,mm,nn	ED,63,mm,nn			ED,73,mm,nn
LD (nm),RR				22,mm,nn	DD,22,mm,nn	FD,22,mm,nn	
LD RR,nm		01,mm,nn	11,mm,nn	21,mm,nn	DD,21,mm,nn	FD,21,mm,nn	31,mm,nn
LD SP,RR				F9	DD,F9	FD,F9	
POP RR	F1	C1	D1	E1	DD,E1	FE,E1	
PUSH RR	F5	C5	D5	E5	DD,E5	FE,E5	
SBC HL,RR		ED,42	ED,52	ED,62			ED,72

ANNEXE I - table 4

Liste des mnémoniques ayant de l'influence sur les quatres indicateurs principaux.

	S	*Z*	*P/V*	*C*
ADC ; ADD A,X ; CP ; EX AF,A ; NEG	x	x	x	x
RL R ; RLC R ; RR R ; RRC R ; SUB	x	x	x	x
SBC ; SLA R ; SRA R ; SRL R	x	x	x	x
AND ; OR ; XOR	x	x	x	0
ADD HL (ou IX ou IY),RR ; CCF				x
RLA ; RLCA ; RRA ; RRCA				x
SCF				1
LDDR ; LDIR			0	
LDD ; LDI			(1)	
LD A,I ; LD A,R	x	x		
BIT		x		
DEC R ou (RR) ; INC R ou (RR) ; IN R,(C)	x	x	x	
CPD ; CPDR ; CPI ; CPIR	x	(1)	(2)	
INDR ; INIR ; OTDR ; OTIR		1		
IND ; INI ; OUTD ; OUTI		(3)		

(1) : P/V = 0 si BC = 0 après exécution ; sinon P/V = 1

(2) : Z = 1 si A = (HL)

(3) : Z = 1 si B = 0 après exécution ; sinon Z = 0

ANNEXE 2

Conversion hexadécimale-décimale et nombres négatifs (déplacement)

h	*d*	*d compl. à 2*	*h*	*d*	*d compl. à 2*	*h*	*d*	*d compl. à 2*	*h*	*d*	*d compl. à 2*
80	128	-128	A0	160	-96	C0	192	-64	E0	224	-32
81	129	-127	A1	161	-95	C1	193	-63	E1	225	-31
82	130	-126	A2	162	-94	C2	194	-62	E2	226	-30
83	131	-125	A3	163	-93	C3	195	-61	E3	227	-29
84	132	-124	A4	164	-92	C4	196	-60	E4	228	-28
85	133	-123	A5	165	-91	C5	197	-59	E5	229	-27
86	134	-122	A6	166	-90	C6	198	-58	E6	230	-26
87	135	-121	A7	167	-89	C7	199	-57	E7	231	-25
88	136	-120	A8	168	-88	C8	200	-56	E8	232	-24
89	137	-119	A9	169	-87	C9	201	-55	E9	233	-23
8A	138	-118	AA	170	-86	CA	202	-54	EA	234	-22
8B	139	-117	AB	171	-85	CB	203	-53	EB	235	-21
8C	140	-116	AC	172	-84	CC	204	-52	EC	236	-20
8D	141	-115	AD	173	-83	CD	205	-51	ED	237	-19
8E	142	-114	AE	174	-82	CE	206	-50	EE	238	-18
8F	143	-113	AF	175	-81	CF	207	-49	EF	239	-17

ANNEXE 2 - Conversion hexadécimale-décimale et nombres négatifs (déplacement) - suite

h	*d*	*d compl. à 2*	*h*	*d*	*d compl. à 2*	*h*	*d*	*d compl. à 2*	*h*	*d*	*d compl. à 2*
90	144	-112	B0	176	-80	D0	208	-48	F0	240	-16
91	145	-111	B1	177	-79	D1	209	-47	F1	241	-15
92	146	-110	B2	178	-78	D2	210	-46	F2	242	-14
93	147	-109	B3	179	-77	D3	211	-45	F3	243	-13
94	148	-108	B4	180	-76	D4	212	-44	F4	244	-12
95	149	-107	B5	181	-75	D5	213	-43	F5	245	-11
96	150	-106	B6	182	-74	D6	214	-42	F6	246	-10
97	151	-105	B7	183	-73	D7	215	-41	F7	247	-9
98	152	-104	B8	184	-72	D8	216	-40	F8	248	-8
99	153	-103	B9	185	-71	D9	217	-39	F9	249	-7
9A	154	-102	BA	186	-70	DA	218	-38	FA	250	-6
9B	155	-101	BB	187	-69	DB	219	-37	FB	251	-5
9C	156	-100	BC	188	-68	DC	220	-36	FC	252	-4
9D	157	-99	BD	189	-67	DD	221	-35	FD	253	-3
9E	158	-98	BE	190	-66	DE	222	-34	FE	254	-2
9F	159	-97	BF	191	-65	DF	223	-33	FF	255	-1

ANNEXE 2

Conversion hexadécimale-décimale (octet haut, octet bas)

h	*d octet bas*	*d octet haut*	*h*	*d*	*h*	*d octet bas*	*d octet haut*	*h*	*d octet bas*	*d octet haut*
00	0	0	20	32	40	64	16384	60	96	24576
01	1	256	21	33	41	65	16640	61	97	24832
02	2	512	22	34	42	66	16896	62	98	25088
03	3	678	23	35	43	67	17152	63	99	25344
04	4	1024	24	36	44	68	17408	64	100	25600
05	5	1280	25	37	45	69	17664	65	101	25856
06	6	1536	26	38	46	70	17920	66	102	26112
07	7	1792	27	39	47	71	18176	67	103	26368
08	8	2048	28	40	48	72	18432	68	104	26624
09	9	2304	29	41	49	73	18688	69	105	26880
0A	10	2560	2A	42	4A	74	18944	6A	106	27136
0B	11	2816	2B	43	4B	75	19200	6B	107	27392
0C	12	3072	2C	44	4C	76	19456	6C	108	27648
0D	13	3328	2D	45	4D	77	19712	6D	109	27904
0E	14	3584	2E	46	4E	78	19968	6E	110	28160
0F	15	3840	2F	47	4F	79	20224	6F	111	28416
10	16	4096	30	48	50	80	20480	70	112	28672

ANNEXE 2 - Conversion hexadécimale-décimale (suite)

h	*d octet bas*	*d octet haut*	*h*	*d*	*h*	*d octet bas*	*d octet haut*	*h*	*d octet bas*	*d octet haut*
11	17	4352	31	49	51	81	20736	71	113	28928
12	18	4608	32	50	52	82	20992	72	114	29184
13	19	4864	33	51	53	83	21248	73	115	29440
14	20	5120	34	52	54	84	21504	74	116	20696
15	21	5376	35	53	55	85	21760	75	117	29952
16	22	5632	36	54	56	86	22016	76	118	30208
17	23	5888	37	55	57	87	22272	77	119	30464
18	24	6144	38	56	58	88	22528	78	120	30720
19	25	6400	39	57	59	89	22784	79	121	30976
1A	26	6656	3A	58	5A	90	23040	7A	122	31232
1B	27	6912	3B	59	5B	91	23296	7B	123	31488
1C	28	7168	3C	60	5C	92	23552	7C	124	21744
1D	29	7424	3D	61	5D	93	23808	7D	125	32000
1E	30	7680	3E	5E	94	94	24064	7E	126	32256
1F	31	7936	3F	63	5F	95	24320	7F	127	32512

L'impression de ce livre
a été réalisée sur les presses
des Imprimeries Aubin
à Poitiers/Ligugé

pour les Éditions PSI

Achevé d'imprimer le 30 mai 1983
N° d'édition, 86595-79-1. — N° d'impression, L 15647
Dépôt légal, juin 1983

ISBN 2-86595-079-4

Imprimé en France

EDITIONS DU P.S.I.
**

82,00 FF

la pratique du ZX Spectrum et du Timex 2000

Cet ouvrage, consacré à la programmation en langage machine, est accessible à tous ceux qui ont assimilé la programmation Basic approfondie. La corrélation entre les commandes Basic et les codes machine a été la préoccupation principale de l'auteur, qui amène progressivement le lecteur à la compréhension de routines de plus en plus complexes. Les trois aspects principaux du Spectrum : le son, la couleur et la haute résolution font l'objet de nombreuses routines dans tous les chapitres. Petit à petit, les opérations logiques, le traitement des boucles, l'affichage particulier du Spectrum et l'animation sur écran sont analysés. Le lecteur attentif élaborera des programmes personnels en code machine quand il aura refermé ce livre.

couverture A.GRAPH

EDITIONS DU P.S.I.
BP 86
France - 77402 lagny s/marne cedex
ISBN 2-86595-079-4
imprimé en France

la pratique du ZX Spectrum

marcel henrot